POLITECNICO DI TORINO

Corso di Laurea Magistrale in Ingegneria Matematica

Tesi di Laurea Magistrale

Gamma processes and their applications

Relatore:
Franco Pellerey

Candidato:
Luca Serri

Contents

Introduction

Speaking about engineering structures and infrastructures, a key problem is the optimization of maintenance, because the amount of money spent for it during the last decades has increased a lot; various mathematical models are being studied to lower the costs [1].

In maintenance, decisions often must be made under uncertainty, and the most important is generally the uncertainty in the deterioration and lifetime of objects. Up to the early nineties, most mathematical maintenance models were based on describing the uncertainty in ageing using a lifetime distribution, but this has a clear disadvantage: it only quantifies whether a component is functioning or not, and thus cannot be used to estimate its deterioration. Another problem is that the failure rate is only useful for making inferences for a large population of components rather than for a single component: in fact, failure rates cannot be observed or measured for a particular component [2].

Chapter 1

Models

For modeling stochastic deterioration, there are various possibilities. One can use a failure rate function or a stochastic process, such as random deterioration rate, Markov process, Brownian motion with drift, non-decreasing jump process (of which the gamma process is a special case).

1.1 Failure rate function

If the lifetime distribution has a cumulative probability distribution $F(t)$ and probability density function $f(t)$, then the failure rate can be defined as

$$r(t) = \frac{f(t)}{1 - F(t)} = \frac{f(t)}{\bar{F}(t)}$$

for $t > 0$. A probabilistic interpretation of it is that $r(t)dt$ represents the probability that a component of age t will fail in the time interval $[t, t + dt]$; for deteriorating elements, the failure rate is an increasing function.

Failure rates have a big disadvantage: they cannot be observed or measured for a particular component [2]. Therefore, a reliability approach solely based on lifetime distributions and their unobservable failure rates is unsatisfactory.

1.2 Random deterioration rate

A very simple approach consists in a stochastic process such that the cumulative amount of deterioration at time t is $X(t) = A\,t$, where A (average deterioration rate) is a random variable with some known distribution, for example uniform or normal. However, this model is not satisfactory when

inspections are involved, since sample paths are straight lines and therefore a single inspection is enough to "remove" the stochastic component of this model [3].

1.3 Markov process

A Markov process is a stochastic process $\{X(t)\}_t$ such that, given the value of $X(t)$, then values of $X(\tau)$ for $\tau > t$ only depend of $X(t)$ and are independent of $X(u)$ where $u < t$; in other words, the conditional distribution is independent of the past and is only determined by the present.

There are various classes of Markov processes that are useful for modeling stochastic deterioration [4]: discrete-time Markov processes with a finite or countable state space (Markov chains), and continuous-time Markov processes with independent increments, such as the Brownian motion with drift and the gamma process. The former is described in the next section, while the latter will be discussed in the next chapter.

Letting $\Delta_{s,t} = X(t) - X(s)$, recall that a stochastic process $\{X(t)\}_t$ has independent increments if and only if Δ_{t_0,t_1} is independent of Δ_{t_2,t_3} for any choice of $t_0 < t_1 \leq t_2 < t_3$. This is more restrictive than the Markovian property: in fact, $X(\tau) = X(t) + \Delta_{t,\tau}$ and with independent increments $\Delta_{t,\tau}$ is independent of $\Delta_{u,t}$ for all $u < t$.

1.3.1 Brownian motion with drift

Recall that a random variable has normal distribution (or gaussian distribution) if its probability density function is

$$\mathcal{N}(x|\mu,\sigma) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

The Brownian motion with drift is a continuous-time stochastic process $\{X(t)\}_{t\geq 0}$ with drift parameter μ and variance parameter σ^2 ($\sigma > 0$) having the following properties:

- $\mathbb{P}(X(0) = 0) = 1$
- $X(t) \sim \mathcal{N}(\mu t, \sigma^2 t)$ for all $t \geq 0$.
- X has independent increments.
- X is continuous.

An example of application of this process is describing the motion of small particles in fluids.

The brownian motion with drift cannot be used for the purpose of a deterioration, since it is not monotone, but has its advantages. For example, explicit expressions can be derived when stress and strength are independent Brownian motions with drift [5].

1.4 Monotonically increasing jump processes

Another possibility consists in modeling the deterioration as a one-dimensional random walk process in which the deterioration steps occur randomly in time at a constant mean rate with step sizes being non-negative, independent, and identically distributed; failure is defined as the event in which the deterioration reaches such a level that the object is considered to be worn out completely [6]. An example consists of studying the lifetime of a device subjected to a sequence of shocks that occur randomly in time: it can be approximated by an homogeneous Poisson process.

In addition to models that consider discrete amounts of damage in isolated instants of time, one can be interested in a model where the deterioration occurs continuously in time. A notable example is gamma process, which is covered in the next chapter.

Chapter 2

Gamma processes

2.1 Definition

Recall that a random variable has gamma distribution with shape parameter $v > 0$ and rate parameter $u > 0$ if its probability density function is

$$Ga(x|v,u) = \frac{u^v}{\Gamma(v)} x^{v-1} e^{-ux} \qquad x > 0$$

where Γ is the gamma function defined by

$$\Gamma(\alpha) = \int_0^{+\infty} z^{\alpha-1} e^{-z} dz$$

It is worth noting that exists another common parameterization which includes shape and scale parameter, with the latter equal to the reciprocal of the rate parameter.

This random variable has expected value and variance

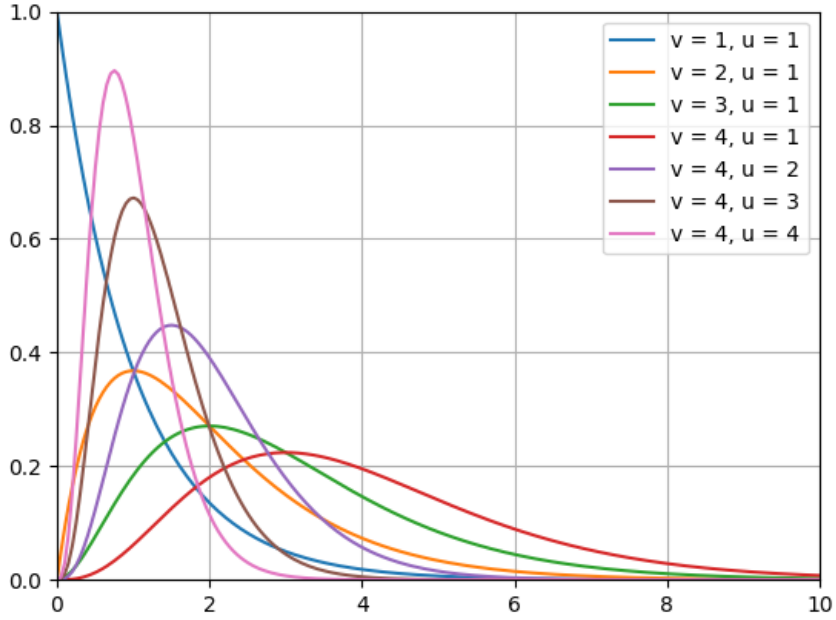
$$E(x|v,u) = \frac{v}{u} \qquad Var(x|v,u) = \frac{v}{u^2}$$

The coefficient of variation is defined by the ratio of the standard deviation and the mean, i.e.

$$c_v(x|v,u) = \frac{\sqrt{Var(x|v,u)}}{E(x|v,u)} = \frac{1}{\sqrt{v}}$$

which is decreasing in time.

The following graph shows the probability density function of gamma-distributed random variables for various values of v and u:



Let $v(t)$ be a non-decreasing, right-continuous, real-valued function for $t \geq 0$ with $v(0) = 0$, $u > 0$, and $\{X(t)\}_{t\geq 0}$ a continuous-time stochastic process with the following properties:

- $\mathbb{P}(X(0) = 0) = 1$
- $X(t_2) - X(t_1) \sim Ga(v(t_2) - v(t_1), u) \;\; \forall t_2 > t_1 \geq 0$
- X has independent increments

Then $\{X(t)\}_{t\geq 0}$ is a gamma process with shape function $v(t)$ and rate parameter u.

2.2 Properties

Let $X(t)$ denote the deterioration at time $t \geq 0$: its probability density function, according to the definition of gamma process, is

$$f_{X(t)} = Ga(x|v(t), u), \tag{2.1}$$

and thus its expectation and variance are

$$E(X(t)) = \frac{v(t)}{u}, \qquad Var(X(t)) = \frac{v(t)}{u^2},$$

while the coefficient of variation is

$$c_v(X(t)) = \frac{\sqrt{Var(X(t))}}{E(X(t))} = \frac{1}{\sqrt{v(t)}},$$

which is decreasing in time, because $v(t)$ is non-decreasing. On the other hand, the ratio of variance and mean is $1/u$ and therefore is constant in time.

Let r_0 denote the initial resistance of a component and $R(t) = r_0 - X(t)$ its deteriorating resistance; the component is said to fail when $R(t)$ falls below a value s, called *stress*. Assuming that both r_0 and s are fixed and known,

$$R(t) < s \qquad \Rightarrow \qquad r_0 - X(t) < s \qquad \Rightarrow \qquad X(t) > r_0 - s = y$$

and we define T_y (lifetime) as the time at which failure occurs. From (2.1) we can obtain the lifetime distribution

$$\begin{aligned} F(t) = \mathbb{P}(T_y \leq t) &= \mathbb{P}(X(t) \geq y) \\ &= \int_y^{+\infty} f_{X(t)}(x)dx \\ &= \int_y^{+\infty} Ga(x|v(t), u)dx \\ &= \int_y^{+\infty} \frac{u^{v(t)}}{\Gamma(v(t))} x^{v(t)-1} e^{-ux} dx \\ &= \frac{1}{\Gamma(v(t))} \int_{yu}^{+\infty} u^{v(t)} \frac{z^{v(t)-1}}{u^{v(t)-1}} e^{-z} \frac{1}{u} dz \\ &= \frac{1}{\Gamma(v(t))} \int_{yu}^{+\infty} z^{v(t)-1} e^{-z} dz \\ &= \frac{\tilde{\Gamma}(v(t), yu)}{\Gamma(v(t))}, \end{aligned}$$

where $\tilde{\Gamma}$ is the incomplete gamma function defined by

$$\tilde{\Gamma}(\alpha, x) = \int_x^{+\infty} z^{\alpha-1} e^{-z} dz.$$

The probability density function can be obtained deriving the lifetime distribution:

$$f(t) = \frac{\partial F(t)}{\partial t} = \frac{\partial}{\partial t} \left(\frac{\tilde{\Gamma}(v(t), yu)}{\Gamma(v(t))} \right)$$

Assuming that the shape function $v(t)$ is differentiable, we can apply the chain rule and obtain

$$f(t) = \frac{\partial}{\partial v(t)} \left(\frac{\tilde{\Gamma}(v(t), yu)}{\Gamma(v(t))} \right) \frac{\partial v(t)}{\partial t}$$

$$= v'(t) \frac{\partial}{\partial v(t)} \left(\frac{\int_{yu}^{+\infty} z^{v(t)-1} e^{-z} dz}{\int_0^{+\infty} z^{v(t)-1} e^{-z} dz} \right)$$

$$\frac{\partial}{\partial v(t)} \int_{yu}^{+\infty} z^{v(t)-1} e^{-z} dz$$

$$= \int_{yu}^{+\infty} \frac{\partial}{\partial v(t)} \left(z^{v(t)-1} e^{-z} \right) dz$$

$$= \int_{yu}^{+\infty} z^{v(t)-1} e^{-z} \log z \, dz,$$

and the same holds for the other integral.

In the following calculations, the integrand will be abbreviated in

$$z^{v(t)-1} e^{-z} \log z = g(v(t), z) = g;$$

recall that the digamma function is defined as

$$\psi(x) = \frac{d \log \Gamma(x)}{dx} = \frac{\Gamma'(x)}{\Gamma(x)}; \tag{2.2}$$

it holds

$$\psi(v(t)) = \frac{\frac{\partial \Gamma(v(t))}{\partial v(t)}}{\Gamma(v(t))} = \frac{\int_0^{+\infty} g \log z \, dz}{\int_0^{+\infty} g \, dz},$$

and therefore

$$f(t) = v'(t) \frac{\partial}{\partial v(t)} \left(\frac{\int_{yu}^{+\infty} g \, dz}{\int_0^{+\infty} g \, dz} \right)$$

$$= v'(t) \left(\frac{\int_{yu}^{+\infty} g \log z \, dz \int_0^{+\infty} g \, dz - \int_{yu}^{+\infty} g \, dz \int_0^{+\infty} g \log z \, dz}{\left(\int_0^{+\infty} g \, dz \right)^2} \right)$$

$$= \frac{v'(t)}{\int_0^{+\infty} g \, dz} \left(\frac{\int_{yu}^{+\infty} g \log z \, dz \int_0^{+\infty} g \, dz - \int_{yu}^{+\infty} g \, dz \int_0^{+\infty} g \log z \, dz}{\int_0^{+\infty} g \, dz} \right)$$

$$= \frac{v'(t)}{\Gamma(v(t))} \left(\int_{yu}^{+\infty} g \log z \, dz - \int_{yu}^{+\infty} g \, dz \frac{\int_0^{+\infty} g \log z \, dz}{\int_0^{+\infty} g \, dz} \right)$$

$$= \frac{v'(t)}{\Gamma(v(t))} \left(\int_{yu}^{+\infty} g \log z \, dz - \int_{yu}^{+\infty} g\psi(v(t)) \, dz \right)$$

$$= \frac{v'(t)}{\Gamma(v(t))} \int_{yu}^{+\infty} (\log z - \psi(v(t))) \, z^{v(t)-1} e^{-z} dz$$

It can be proved that if $v'(t) > 0$, then the failure rate is increasing.

If fixed values of initial resistance and stress are not satisfactory, one can opt for a different model where y is replaced by a random quantity $Y > 0$. Given the probability density function $f_Y(y)$, the probability of failure in time interval $(0, t)$ is

$$\mathbb{P}(X(t) > Y) = \int_0^{+\infty} f_{X(t)}(x) \mathbb{P}(Y \leq x) dx$$

$$= \int_0^{+\infty} \int_0^x f_{X(t)}(x) f_Y(y) dy dx.$$

The probability $\mathbb{P}(Y > x) = 1 - G(x) = \bar{G}(x)$ can be interpreted as the probability that a device survives x units of deterioration. It can be shown that the cumulative distribution function has an increasing failure rate if $v(t)$ is convex and $G(x)$ has an increasing failure rate.

2.3 Expected deterioration

When modeling the temporal variability in the deterioration, a crucial question is how the expected deterioration increases over time. Empirical studies show that the expected deterioration at time t is often proportional to a power law:

$$E(X(t)) = \frac{v(t)}{u} \sim \frac{c\, t^b}{u} = a\, t^b \tag{2.3}$$

Examples of expected deterioration according to a power law include the expected degradation of concrete due to corrosion of reinforcement ($b = 1$), sulphate attack ($b = 2$), diffusion-controlled ageing ($b = 0.5$) [7]. In the special case $b = 1$ the expected deterioration is linear in time, and the gamma process is called stationary.

These cases are notable because estimating the time of failure is simple: in fact, given critical value of deterioration y then

$$y = a\, t_{fail}^b \qquad \Rightarrow \qquad t_{fail} = \sqrt[b]{\frac{y}{a}}$$

2.4 Parameter estimation

A typical dataset contains inspection times $\{t_j\}_{j=1,...,n}$ (with $0 = t_0 < t_1 < ... < t_n$) and corresponding observations of cumulative amounts of deteriorations $\{x_j\}_{j=1,...,n}$ (with $0 = x_0 \leq x_1 \leq ... \leq x_n$), often taken for multiple objects. Given this data and supposing that the expected deterioration is (2.3), the two most common methods of parameter estimation are method of maximum likelihood and method of moments.

2.4.1 Method of maximum likelihood

The maximum likelihood estimators of c and u can be obtained by maximizing the logarithm of the likelihood function of the increments. Letting $w_j = t_j^b - t_{j-1}^b$, the likelihood function of the observed deterioration increments $\delta_j = x_j - x_{j-1}$ is a product of independent gamma densities

$$\ell(\delta_1, ..., \delta_n | c, u) = \prod_{j=1}^{n} f_{X(t_j)-X(t_j-1)}(\delta_j) = \prod_{j=1}^{n} \frac{u^{cw_j}}{\Gamma(cw_j)} \delta_j^{cw_j - 1} e^{-u\delta_j} \tag{2.4}$$

By computing the first partial derivatives of the loglikelihood function of the increments with respect to c and u, the maximum likelihood estimates $\hat{c}$ and $\hat{u}$ can be computed by solving the following system (see Appendix for calculations):

$$\begin{cases} \hat{u} = \dfrac{\hat{c} t_n^b}{x_n} \\ \displaystyle\sum_{j=1}^{n} w_j(\psi(\hat{c} w_j) - log\delta_j) = t_n^b \log(\hat{u}) \end{cases} \tag{2.5}$$

From the first equation it follows that the expected deterioration at time t can be written as

$$E(X(t)) = x_n \left(\frac{t}{t_n} \right)^b$$

Because cumulative amounts of deterioration are measured, the last inspection contains the most information; this is confirmed by the fact that the expected deterioration at the last inspection time t_n equals x_n, i.e. $E(X(t_n)) = x_n$.

This method can be used to estimate b as well: this parameter can be obtained by maximizing the likelihood function (2.4). In particular, Nicolai et al. [9] applied this approach to fit a gamma process to inspection data of the Dutch Haringvliet storm-surge barrier representing percentages of steel-gate surfaces that have been corroded due to ageing of the coating.

2.4.2 Method of moments

Given that the expected value and the variance of cumulative deterioration at time t are

$$E(X(t)) = \frac{ct^b}{u}, \qquad Var(X(t)) = \frac{ct^b}{u^2},$$

when the power b is known the gamma process can be transformed into a stationary gamma process with a monotonic transformation $t \mapsto z(t) = t^b$ obtaining

$$E(X(z) = \frac{cz}{u}, \qquad Var(X(z) = \frac{cz}{u^2},$$

which is a stationary gamma process with respect to the transformed time z. According to Çinlar et al. [8] the extimates can be computed by solving the system

$$\begin{cases} \dfrac{\hat{c}}{\hat{u}} = \dfrac{\sum_{j=1}^{n} \delta_j}{\sum_{j=1}^{n} w_j} = \dfrac{x_n}{t_n^b} = \bar{\delta} \\ \dfrac{x_n}{\hat{u}} \left(1 - \dfrac{\sum_{j=1}^{n} w_j^2}{(\sum_{j=1}^{n} w_j)^2} \right) = \displaystyle\sum_{j=1}^{n} (\delta_j - \bar{\delta} w_j)^2 \end{cases} \tag{2.6}$$

The first equation is the same as (2.5) but the second one is simpler, because it can be rearranged to obtain an explicit expression for parameter $\hat{u}$, so parameter extimation is easier.

Chapter 3

Implementation and testing

Gamma processes can be implemented in code and used for various purposes. In this study Python language is used [10], with libraries NumPy [11], SciPy [12] and Matplotlib [13]. The code is written by myself and is available at the following link: `https://github.com/lucas992x/gammaprocesses`

At first, formulas described in the previous chapters are used to generate random samples, compute the parameters using this data and compare obtained values with initial values. Inspection times and parameters b, c, u are arbitrarily chosen; a critical value is passed too, to analyze failure times and try to approximate them with some known distribution.

After this, the code is applied to two case studies, which are illustrated in the next chapter.

3.1 Implementation

3.1.1 Generating random samples

From the definition of gamma process given in Section 2.1 it follows that

$$X(t_j) - X(t_{j-1}) \sim Ga(c(t_j^b - t_{j-1}^b), u)$$

for each couple of inspection times $0 \leq t_{j-1} < t_j$. This equation is used to generate random degradation paths with `numpy.random.gamma`; however, NumPy expects shape and scale parameters while formulas described in the previous chapter (and therefore this equation) are based on shape and rate parameters, so the distribution is rewritten as

$$X(t_j) - X(t_{j-1}) \sim Ga\left(c(t_j^b - t_{j-1}^b), \frac{1}{u}\right)$$

Given the number of samples needed (`num`), the list of inspection times (`times`) and the parameters (`b`, `c`, `u`), the code generates random deterioration increments and sums them to obtain cumulative deterioration in inspection times.

```
def GenerateSamples(num, times, b, c, u):
    samples = []
    for j in range(num):
        sample = [0]
        for j in range(1, len(times)):
            incr = np.random.gamma(c * (times[j] ** b - times
                                        [j - 1] ** b), 1 /
                                         u)
            sample.append(incr + sample[-1])
        samples.append(sample)
    return samples
```

It is possible to pass a critical value, that states whether objects are considered to have failed or not, in that case this value is added to the title:

```
def PrintPlotSamples(t, samples, b, c, u, where, method =
                               None, limits = [[], []],
                               critical = 0):
    if method is None:
        title = 'Original samples'
    else:
        title = 'Samples generated with {}\nb = {:.3f}, c = {
                                      :.3f}, u = {:.3f}, a =
                                       {:.3f}'.format(method
                                      , b, c, u, c / u)
    if critical > 0:
        title += ', critical = {}'.format(critical)
    [...]
```

The user can decide to print the samples to console, plot them on a graph, or both:

```
def PrintSample(sample, sep, pad, decs):
    print(sep.join(['{:{}.{}f}'.format(s, pad + 1 + decs,
                                    decs) for s in sample]).
                                    strip())

def PrintPlotSamples(t, samples, b, c, u, where, method =
                               None, limits = [[], []],
                               critical = 0):
    [...]
    # print to console
    if where in ['console', 'both']:
        maxvalue = max(t[-1], max([v[-1] for v in samples]))
        decs = 3
```

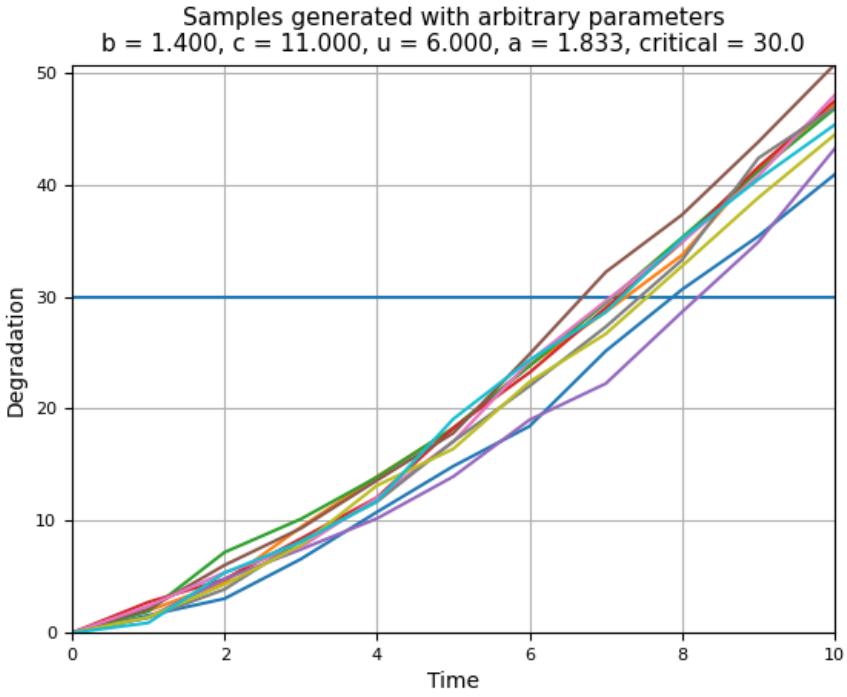
```
        pad = len('{:.{}f}'.format(maxvalue, decs)) - 1 -
                                            decs
        print('\n{}:'.format(title))
        PrintSample(t, args.sep, pad, decs)
        for sample in samples:
            PrintSample(sample, args.sep, pad, decs)
    # plot to graph
    if where in ['graphs', 'both']:
        PlotGraph(t, samples, title, limits[0], limits[1],
                                            critical)
```

The graph is plotted using `matplotlib.pyplot`. If argument `yy` contains exactly two elements, the code prints the estimated pdf of failure times (see next section), otherwise it prints all samples that were generated with arbitrary parameters passed by the user. In both cases the graph is automatically saved as image.

```
def PlotGraph(x, yy, title, xlimits = [], ylimits = [],
                                critical = 0, labels = ['Time'
                                , 'Degradation']):
    # adjust text size
    rcparams = {
    'axes.titlesize': 11,
    'axes.labelsize': 10,
    'xtick.labelsize': 8,
    'ytick.labelsize': 8,
    }
    plt.rcParams.update(rcparams)
    if len(yy) == 2:
        # in this case plot normal pdf with a dotted line and
                                        estimated pdf with a
                                        continuous line
        plt.plot(x, yy[1], linestyle = ':')
        plt.plot(x, yy[0])
    else:
        # plot horizontal line with critical level
        if critical > 0:
            plt.axhline(critical)
        # plot all samples
        for y in yy:
            plt.plot(x, y)
    plt.grid()
    plt.xlabel(labels[0])
    plt.ylabel(labels[1])
    if xlimits:
        plt.xlim(*xlimits)
    if ylimits:
        plt.ylim(*ylimits)
    plt.title(title)
```

```
    plt.savefig('{}.png'.format(title.replace('\n', ' ')))
    plt.close()
```

Here is an example of a graph with random degradation paths and a critical value:



3.1.2 Computing parameters

Method of moments is very straightforward to apply, because equations (2.6) can be easily explicited to obtain c and u:

```
def SolveMoments(t, x, b):
    n = len(t)
    delta = [x[j] - x[j - 1] for j in range(1, n)]
    deltabar = x[-1] / (t[-1] ** b)
    w = [t[j] ** b - t[j - 1] ** b for j in range(1, n)]
    u = x[-1] * (1 - sum([wj ** 2 for wj in w]) / (sum(w) **
                                    2)) / sum([(delta[j] -
                                    deltabar * w[j]) ** 2 for
                                    j in range(n - 1)])
    c = u * deltabar
    return c, u
```

Instead, method of maximum likelihood is more complicated because (2.5) involves a minimization with two constrains (2.4).

```
# likelihood function of the increments with respect to c and
                                    u
def IncrLikelihood(params, *args):
    b, c, u = params
    t, delta = args
    w = [t[j] ** b - t[j - 1] ** b for j in range(1, len(t))]
    # to maximize f, minimize -f
    return -np.prod([(u ** (c * w[j])) * (delta[j] ** (c * w[
                                    j] - 1)) * np.exp(-1 * u *
                                     delta[j]) / (gamma(c * w[
                                    j])) for j in range(len(w)
                                    )])

# maximum-likelihood estimator of c (used as constraint to
                                    solve using ML method)
def MaxLikeC(params, *args):
    b, c, u = params  # u unused here
    t, delta, xn = args
    w = [t[j] ** b - t[j - 1] ** b for j in range(1, len(t))]
    logarg = c * (t[-1] ** b) / xn
    # avoid errors if logarithm argument is not positive
    if logarg <= 0:
        # any non-zero value is ok to make the constraint
                                        unsatisfied
        return 666
    else:
        return sum([w[j] * (digamma(c * w[j]) - np.log(delta[
                                        j])) for j in range(
                                        len(w))]) - (t[-1] **
                                        b) * np.log(logarg)

# maximum-likelihood estimator of u (used as constraint to
                                    solve using ML method)
def MaxLikeU(params, *args):
    b, c, u = params
    tn, xn = args
    return u - c * (tn ** b) / xn
```

To solve this, `scipy.optimize.minimize` is used when b is unknown. The function returns 0 if the minimization fails.

```
# solve using method of maximum likelihood
# if guesses = [b0, c0, u0] all parameters are evaluated,
                                    with this values as initial
                                    guesses
# if guesses = [b0, c0] only c and u are evaluated, b0 is
                                    fixed and c0 initial guess
def SolveMaxLike(t, x, guesses):
    n = len(x)
```

```
    delta = [x[j] - x[j - 1] for j in range(1, n)]
    if len(guesses) == 3:
        mins = minimize(IncrLikelihood, guesses, args = (t,
                                        delta), constraints =
                                        [
        {'type': 'eq', 'fun': MaxLikeC, 'args': (t, delta, x[
                                        -1])},
        {'type': 'eq', 'fun': MaxLikeU, 'args': (t[-1], x[-1]
                                        )} ],
        bounds = ((0, None), (0, None), (0, None)))
        if mins.success == True:
            return mins.x[0], mins.x[1], mins.x[2]  # b, c, u
        else:
            return 0, 0, 0
    else:
        [...]
```

When b is known, the code uses (2.5) to compute c and u: the first equation is solved using `scipy.optimize.fsolve`.

```
# solve using method of maximum likelihood
# if guesses = [b0, c0, u0] all parameters are evaluated,
                            with this values as initial
                            guesses
# if guesses = [b0, c0] only c and u are evaluated, b0 is
                            fixed and c0 initial guess
def SolveMaxLike(t, x, guesses):
    n = len(x)
    delta = [x[j] - x[j - 1] for j in range(1, n)]
    if len(guesses) == 3:
        [...]
    else:
        b0, c0 = guesses
        w = [t[j] ** b0 - t[j - 1] ** b0 for j in range(1, n)
                                        ]
        func = lambda c : sum([w[j] * (digamma(c * w[j]) - np
                                        .log(delta[j])) for j
                                        in range(n - 1)]) - (t
                                        [-1] ** b0) * np.log(c
                                         * (t[-1] ** b0) / x[-
                                        1])
        c = fsolve(func, c0)[0]
        u = c * (t[-1] ** b0) / x[-1]
        return c, u
```

Both `scipy.optimize.minimize` and `scipy.optimize.fsolve` require an initial guess, i.e. a vector of values "close" to the solution.

In the first case, an approximation of b is obtained by fitting the function $f(t) = a\,t^b$ (expected deterioration) with `scipy.optimize.curve_fit`, while

approximations of c and u are calculated using explicit formulas of method of moments (2.6) using fitted b. If this does not lead to a solution, the code tries with various guesses for b and then chooses the computed value that maximizes the likelihood function.

In the second case only c needs to be approximated, and this is done using method of moments as in the first case. This method is also used to solve with method of maximum likelihood using (2.5) with fitted b, to compare the results.

There is no need to explicitly specify which case is being used: the code uses the right one by checking how many guesses are being passed as arguments.

```
def Expon(t, a, b):
    return a * (t ** b)

def SolveAll3(t, xx, bguesses, percentiles, prnt = False):
    [...]
    for x in xx:
        # fit function
        params = curve_fit(Expon, t, x)
        a0, b0 = params[0]
        c0, u0 = SolveMoments(t, x, b0)  # also used as
                                         initial guesses that
                                         satisfy constrains
        # solve with method of maximum likelihood
        b1, c1, u1 = SolveMaxLike(t, x, [b0, c0, u0])
        if b1 > 0:
            solML = [b1, c1, u1]
        else:
            # try to solve with various guesses for b
            solutions = []
            for bg in bguesses:
                c0, u0 = SolveMoments(t, x, bg)  # get
                                                   initial
                                                   guesses for c
                                                   and u that
                                                   satisfy
                                                   constrains
                b1, c1, u1 = SolveMaxLike(t, x, [bg, c0, u0])
                if b1 > 0:
                    solutions.append([b1, c1, u1])
            if solutions == []:
                solML = [0, 0, 0]
            else:
                # find the actual minimum among solutions (
                                                   which can
                                                   contain local
```

```
                                                minima)
                delta = [x[j] - x[j - 1] for j in range(1,
                                                len(x))]
                minimum = sys.float_info.max
                solML = 3 * [minimum]
                for solution in solutions:
                    llvalue = IncrLikelihood(solution, t,
                                                    delta)
                    if llvalue < minimum:
                        minimum = llvalue
                        solML = solution
        c2, u2 = SolveMaxLike(t, x, [b0, c0])
        c3, u3 = SolveMoments(t, x, solML[0])
        [...]
```

When multiple samples are available, the code computes parameters for all of them and then calculates mean, variance, standard deviation and two percentiles. If there are more than 10 samples the code removes some of them, specifically lowest and greatest values, in a quantity equal to the lowest between 10% of samples and 5 samples; this is done because the mean is not very stable, in the sense that even a single value can drastically change its value if it is significantly greater or lower than the others.

```
# compute mean, variance, standard deviation and two
                                percentiles of some data (
                                default 2.5\% and 97.5\%)
class Stats:
    def __init__(self, data, percentile = 2.5):
        # remove some data to avoid a result "ruined" by a
                                        few mistakes
        if len(data) >= 10:
            data = sorted(data)
            remove = min([len(data) // 10, 5])
            self.values = data[remove:-remove]
        else:
            self.values = data
        self.n = len(self.values)
        self.mean = np.mean(self.values)
        sqsum = sum([(x - self.mean) ** 2 for x in self.
                                        values])
        self.variance = sqsum / self.n
        self.std = np.sqrt(sqsum / (self.n - 1))
        self.perc = percentile
        self.lowperc = np.percentile(self.values, percentile)
        self.upperc = np.percentile(self.values, 100 -
                                        percentile)
```

After computing parameters with these methods, they are compared using

Akaike Information criterion [14] and Bayesian information criterion [14], two criterions commonly used to choose a model among a finite set. They are based on the same idea: since the likelihood of a model can be increased by adding parameters, they add a penalty term that increases when more parameters are used, which also helps to prevent overfitting.

Letting k be the number of estimated parameters, n the number of samples and $\hat{L}$ the maximum value of the likelihood function, they are defined as

$$AIC = 2k - 2\log \hat{L} \tag{3.1}$$

$$BIC = k\log n - 2\log \hat{L} \tag{3.2}$$

In both cases the model with lowest value is preferred; this is intuitive, because the goal is to maximize the likelihood while preventing the model from becoming too complex.

Implementation in code is simple and straightforward: `IncrLikelihood` function was already defined before, here is used to compute the likelihood of entire dataset, recalling that $\log xy = \log x + \log y$.

```
def CalcCriterions(b, c, u, t, xx, bknown = False, method = '
                                    '):
    loglike = 0  # loglikelihood
    for x in xx:
        delta = [x[j] - x[j - 1] for j in range(1, len(x))]
        loglike += np.log(-IncrLikelihood([b, c, u], t, delta
                                          ))
    if bknown == True:
        k = 2
    else:
        k = 3
    aic = 2 * (k - loglike)
    bic = k * np.log(len(xx)) - 2 * loglike
    if method:
        print('{}  {:8.3f}  {:8.3f}'.format(method, aic, bic)
                                          )
    return aic, bic
```

3.1.3 Time of failure estimation

If a critical value is passed, the code computes the estimated probability density function of failure time. Intuitively, this p.d.f. should assume higher values in regions where there is more data.

Recall that a window function is a function that is null outside a given

interval; a very simple window function is

$$\varphi(x) = \begin{cases} 1 & |x| \leq \frac{1}{2} \\ 0 & otherwise \end{cases}$$

which is null outside $[-\frac{1}{2}, \frac{1}{2}]$ and assumes value 1 inside this interval. This function is also a density, because it is non-negative over $\mathbb{R}$ and it is easy to check that $\int_{\mathbb{R}} \varphi(x)dx = 1$. It can be adapted to any interval: in fact, given $[a, b]$ then

$$\varphi_{a,b}(x) = \frac{1}{b-a}\varphi\left(\frac{x - \frac{a+b}{2}}{b-a}\right)$$

assumes value $\frac{1}{b-a}$ inside $[a, b]$ and 0 outside, which means it's still a density.

Given n samples and an arbitrary value $h > 0$, one can compute this on intervals $[x_j - h, x_j + h]$ for each sample x_j. The function

$$f_h(x) = \frac{1}{n}\sum_{j=1}^{n} \frac{1}{h}\varphi\left(\frac{x - x_j}{h}\right)$$

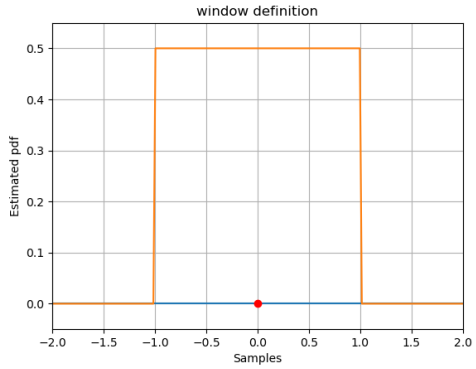
is a density, in fact it is non-negative (because φ is non-negative by definition and h is positive) and

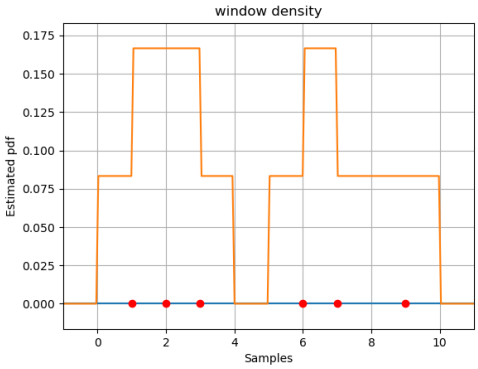
$$\int_{-\infty}^{+\infty} f_h(x)dx = \int_{-\infty}^{+\infty} \frac{1}{n}\sum_{j=1}^{n} \frac{1}{h}\varphi\left(\frac{x - x_j}{h}\right) dx$$

$$= \frac{1}{n}\sum_{j=1}^{n} \frac{1}{h}\int_{-\infty}^{+\infty} \varphi\left(\frac{x - x_j}{h}\right) dx$$

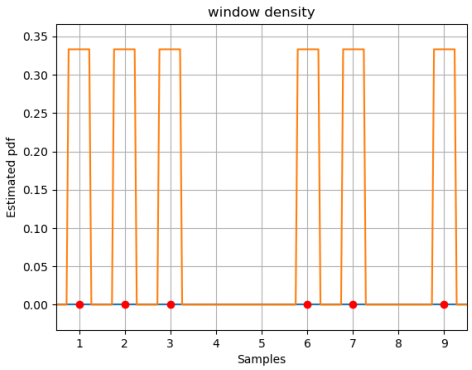
$$= \frac{1}{n}\sum_{j=1}^{n} \frac{1}{h}h = 1$$

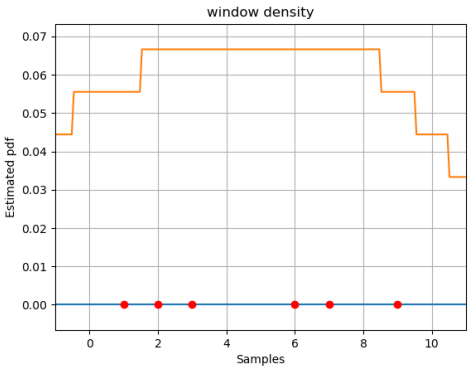
The following graphs show the window function for a single sample and the resulting density function obtained with samples $\{1, 2, 3, 6, 7, 9\}$ and $h = 2$:





It is important to note that not all h give good results. These two graphs show what happens with h too small or too large:



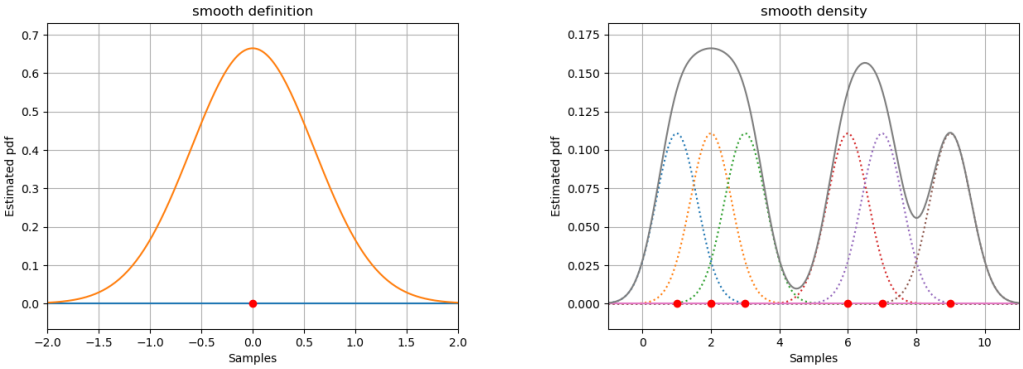


With small h each window contains only one sample, and therefore the function will assume value $\frac{1}{nh}$ near a sample and 0 otherwise. With large h the function becomes nearly constant over considered samples.

However, the resulting density is not smooth and has discontinuities. A common choice to make the density smooth is the p.d.f. of a normal distribution with zero mean, i.e.

$$N_{0,\sigma}(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{x^2}{2\sigma^2}}$$

It is worth noting that this is not a window function, although its value is very close to zero outside an interval. σ should be chosen properly, for the same reason of h; in this case, with $\sigma = 0.3$ the result looks good:



These functions are called kernel functions.

Estimating the failure time for a sample is pretty straightforward: the code finds the last inspection before failure and the first inspection after it, and applies a linear proportion to estimate the time of failure.

```
def GetFailureTime(t, x, critical):
    # failure has not happened yet
    if x[-1] < critical:
        failtime = None
    # time of failure is contained in t
    elif critical in x:
        failtime = t[x.index(critical)]
    # estimate time of failure
    else:
        index = x.index(max([xx for xx in x if xx < critical]
                                    ))  # index of last
                                    inspection before
                                    failure
        prev = [t[index], x[index]]  # last inspection before
                                     failure
        next = [t[index + 1], x[index + 1]]  # first
                                    inspection after
                                    failure
        # apply a proportion to compute time of failure
        failtime = prev[0] + (next[0] - prev[0]) * (critical
                                    - prev[1]) / (next[1]
                                    - prev[1])
    return failtime
```

Once failure times are all computed, the code estimates the probability density function using Gaussian kernels (`scipy.stats.gaussian_kde`) and plots it. The p.d.f. of a normal distribution with mean and variance equal to sample mean and sample variance is evaluated and plotted too (using `scipy.stats.norm.pdf`), to compare the two functions:

```
def GetFailurePdf(t, samples, critical, b = None, c = None, u
                                = None, percentile = 2.5):
    # compute failure times
    failuretimes = []
    for sample in samples:
        failuretime = GetFailureTime(t, sample, critical)
        if failuretime is not None:
            failuretimes.append(failuretime)
    # if no sample reached failure there's no need to compute
                                    pdf
    if failuretimes == []:
        print('No sample reached failure!')
    # same when only one sample reached failure
    elif len(failuretimes) == 1:
        print('Only one sample reached failure, at time {:.3f
                                    }'.format(failuretimes
                                    [0]))
    else:
        failuretimes = Stats(failuretimes, percentile =
                                    percentile)
        # plot the graph
        xgraph = np.linspace(0.9 * failuretimes.lowperc, 1.1
                                    * failuretimes.upperc,
                                     num = 100)
        title = 'Estimated pdf of failure time with {}
                                    samples'.format(len(
                                    samples))
        # parameters will be printed if arbitrary
        if b is not None:
            title += '\nb = {:.3f}, c = {:.3f}, u = {:.3f}, a
                                        = {:.3f}'.format(
                                       b, c, u, c / u)
        title += ', critical = {}'.format(critical)
        # estimate pdf of failure time using Gaussian kernels
        kde = gaussian_kde(failuretimes.values)
        # normal distribution pdf, to compare it with
                                    estimated pdf
        normpdf = norm.pdf(xgraph, failuretimes.mean,
                                    failuretimes.std)
        # plot the two curves on the same graph
        PlotGraph(xgraph, [kde(xgraph), normpdf], title, [min
                                    (xgraph), max(xgraph)]
                                    , [0, 1.05 * max([max(
                                    kde(xgraph)), max(
                                    normpdf)])], labels =
                                    ['', ''])
```

3.2 Testing the code

3.2.1 Parameter estimation

Random samples are generated using three sets of arbitrary parameters, recalling that $a = c/u$:

b	c	u	a
1	3.14	2.72	1.154
1.4	11	6	1.833
2	4	8	0.5

The first method used to estimate parameters is simple and straightforward: since the expected deterioration is often in the form $E(X(t)) = a\,t^b$, the code tries to fit this function to estimate a and b. Here are the results:

Parameter	Value	Mean	Std	$p_{0.025}$	$p_{0.975}$
b	1	1.032	0.175	0.718	1.409
a	1.154	1.174	0.500	0.430	2.336
b	1.4	1.401	0.088	1.233	1.575
a	1.833	1.871	0.389	1.207	2.713
b	2	2.007	0.103	1.812	2.206
a	0.500	0.509	0.124	0.306	0.780

b is very accurate in all cases, and estimation of a is good too.

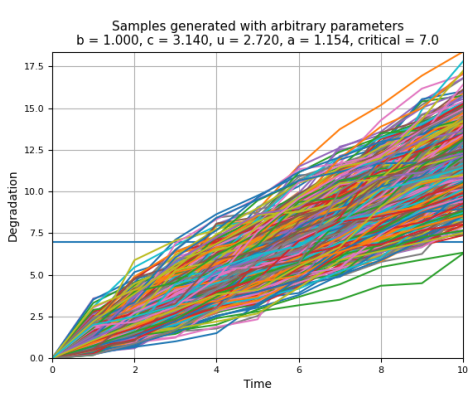
The second method is maximum likelihood, described in Section 2.4.1, which estimates b, c and u. Here are the results with the same arbitrary parameters:

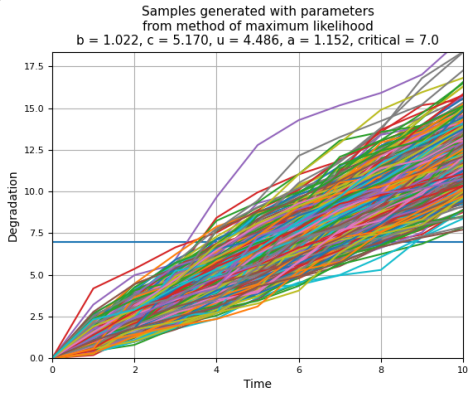
Parameter	Value	Mean	Std	$p_{0.025}$	$p_{0.975}$
b	1	1.022	0.182	0.683	1.403
c	3.14	5.170	3.337	1.349	13.909
u	2.72	4.486	2.663	1.738	11.514
b	1.4	1.432	0.098	1.250	1.633
c	11	15.007	8.042	4.524	36.732
u	6	8.581	4.008	3.215	18.647
b	2	1.957	0.118	1.736	2.200
c	4	5.265	2.224	2.161	10.436
u	8	9.288	2.973	4.070	14.445

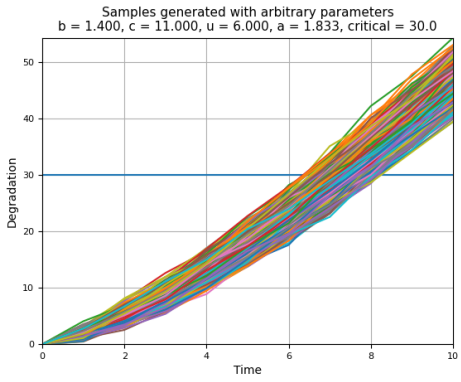
b is still very accurate, while c and u are significantly different. At first glance one can think that this implies a bad model; however, it is worth noting that the estimation of a is accurate:

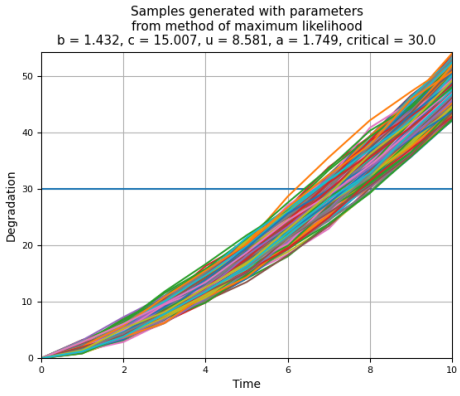
a	Estimated
1.154	1.152
1.833	1.749
0.500	0.567

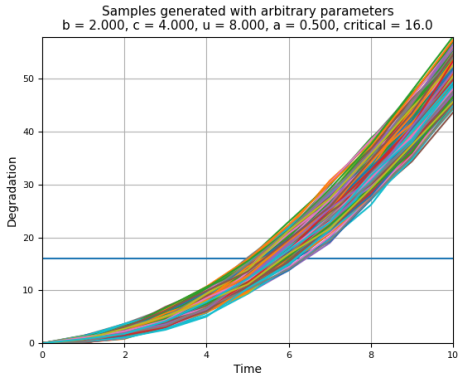
From (2.3) it follows that the expected deterioration is unchanged if c and u are modified keeping c/u constant; it means that these parameters can still be used to approximate the expected deterioration. This is evident when looking at the graphs, the following pictures show a comparison between random samples generated with input parameters (left) and values obtained from method of maximum likelihood (right):

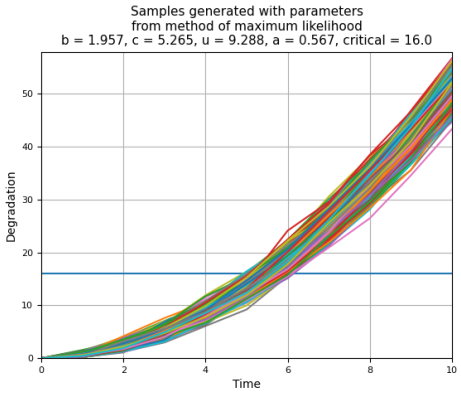








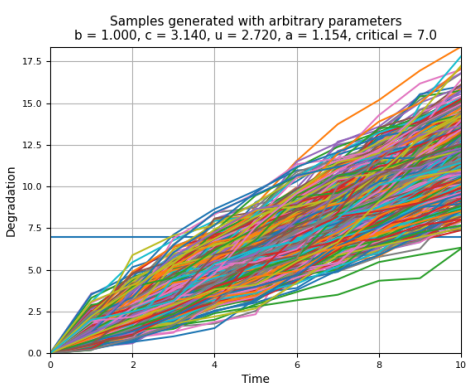


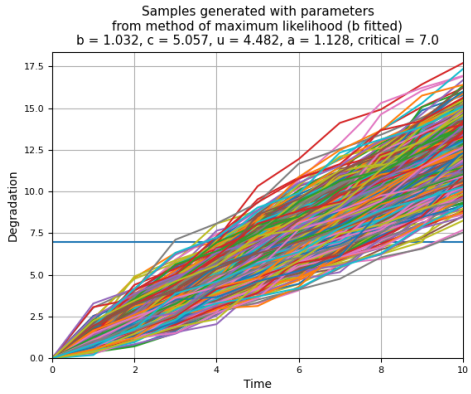


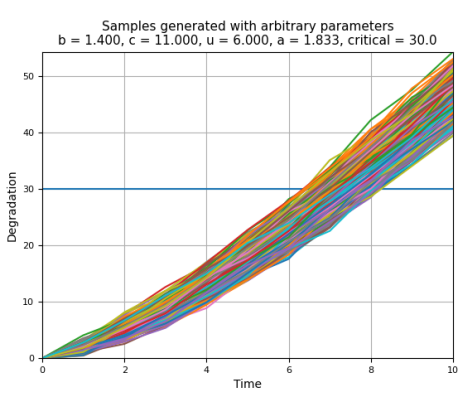
Similar results are obtained when computing c and u with method of maximum likelihood but using fitted value of b:

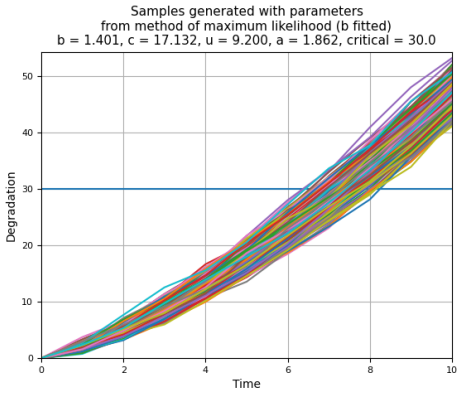
Parameter	Value	Mean	Std	$p_{0.025}$	$p_{0.975}$
c	3.14	5.057	3.248	1.328	14.068
u	2.72	4.482	2.599	1.792	11.484
c	11	17.132	10.205	5.396	41.856
u	6	9.200	4.904	3.197	21.238
c	4	6.170	3.639	2.165	17.296
u	8	12.488	7.492	4.567	33.399

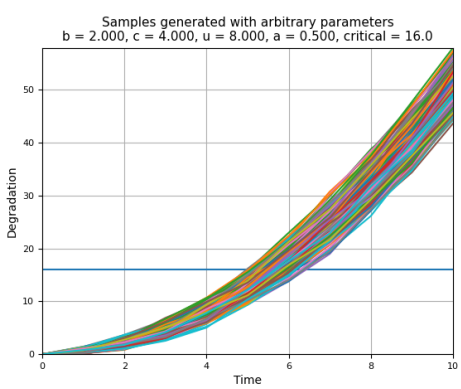
a	Estimated
1.154	1.128
1.833	1.862
0.500	0.494

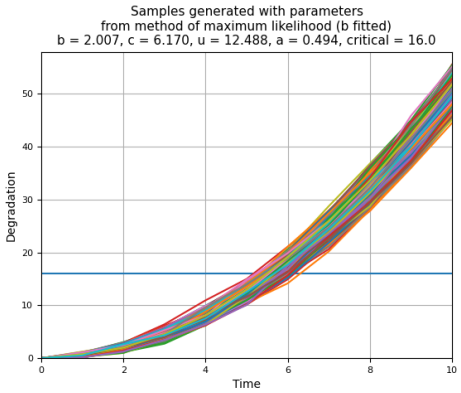












As seen in Section 2.4.2, method of moments can only estimate c and u when b is assumed to be known. Since b can be estimated wth the two methods described above, they will be combined with method of moments to compare results. In both cases, similar considerations hold: a is accurate although c and u are not, which means that this method is still good to estimate the expected deterioration.

The following tables show a comparison of values obtained from all methods discussed above:

First set of arbitrary parameters

Parameter	Value	Mean	Std	$p_{0.025}$	$p_{0.975}$
b (fitted)	1	1.032	0.175	0.718	1.409
a (fitted)	1.154	1.174	0.500	0.430	2.336
b (method of maximum likelihood)	1	1.022	0.182	0.683	1.403
c (method of maximum likelihood)	3.14	5.170	3.337	1.349	13.909
u (method of maximum likelihood)	2.72	4.486	2.663	1.738	11.514
a (method of maximum likelihood)	1.154	1.152			
c (method of maximum likelihood, b fitted)	3.14	5.057	3.248	1.328	14.068
u (method of maximum likelihood, b fitted)	2.72	4.482	2.599	1.792	11.484
a (method of maximum likelihood, b fitted)	1.154	1.128			
c (method of moments, b fitted)	3.14	4.889	3.260	1.105	13.509
u (method of moments, b fitted)	2.72	4.281	2.460	1.512	11.108
a (method of moments, b fitted)	1.154	1.142			
c (method of moments, b from ML)	3.14	4.981	3.380	1.091	13.870
u (method of moments, b from ML)	2.72	4.235	2.505	1.394	11.382
a (method of moments, b from ML)	1.154	1.176			

First set of arbitrary parameters

Method	AIC	BIC
Method of maximum likelihood	9451.631	9464.275
Method of maximum likelihood (b fitted)	9464.577	9477.221
Method of moments (b fitted)	9380.101	9392.744
Method of moments (b from ML)	9379.644	9392.288

In this case both AIC and BIC suggest that method of moments is better, with b computed from maximum likelihood method slightly better than fitted b.

Second set of arbitrary parameters

Parameter	Value	Mean	Std	$p_{0.025}$	$p_{0.975}$
b (fitted)	1.4	1.401	0.088	1.233	1.575
a (fitted)	1.833	1.871	0.389	1.207	2.713
b (method of maximum likelihood)	1.4	1.432	0.098	1.250	1.633
c (method of maximum likelihood)	11	15.007	8.042	4.524	36.732
u (method of maximum likelihood)	6	8.581	4.008	3.215	18.647
a (method of maximum likelihood)	1.833	1.749			
c (method of maximum likelihood, b fitted)	11	17.132	10.205	5.396	41.856
u (method of maximum likelihood, b fitted)	6	9.200	4.904	3.197	21.238
a (method of maximum likelihood, b fitted)	1.833	1.862			
c (method of moments, b fitted)	11	15.408	9.305	4.837	41.217
u (method of moments, b fitted)	6	8.267	4.473	2.866	20.463
a (method of moments, b fitted)	1.833	1.864			
c (method of moments, b from ML)	11	13.776	7.890	4.001	35.074
u (method of moments, b from ML)	6	7.811	3.887	2.773	17.500
a (method of moments, b from ML)	1.833	1.764			

Second set of arbitrary parameters

Method	AIC	BIC
Method of maximum likelihood	13218.099	13230.743
Method of maximum likelihood (b fitted)	13242.823	13255.467
Method of moments (b fitted)	12972.048	12984.691
Method of moments (b from ML)	13103.211	13115.854

Method of moments is again the winner, but in this case fitted b is significantly better than b computed with method of maximum likelihood.

Third set of arbitrary parameters

Parameter	Value	Mean	Std	$p_{0.025}$	$p_{0.975}$
b (fitted)	2	2.007	0.103	1.812	2.206
a (fitted)	0.500	0.509	0.124	0.306	0.780
b (method of maximum likelihood)	2	1.957	0.118	1.736	2.200
c (method of maximum likelihood)	4	5.265	2.224	2.161	10.436
u (method of maximum likelihood)	8	9.288	2.973	4.070	14.445
a (method of maximum likelihood)	0.500	0.567			
c (method of maximum likelihood, b fitted)	4	6.170	3.639	2.165	17.296
u (method of maximum likelihood, b fitted)	8	12.488	7.492	4.567	33.399
a (method of maximum likelihood, b fitted)	0.500	0.494			
c (method of moments, b fitted)	4	5.819	3.817	1.722	15.980
u (method of moments, b fitted)	8	11.728	7.847	3.604	32.545
a (method of moments, b fitted)	0.500	0.496			
c (method of moments, b from ML)	4	5.165	2.667	1.831	11.811
u (method of moments, b from ML)	8	9.068	3.802	3.419	18.490
a (method of moments, b from ML)	0.500	0.570			

Third set of arbitrary parameters

Method	AIC	BIC
Method of maximum likelihood	10884.749	10897.393
Method of maximum likelihood (b fitted)	11188.894	11201.538
Method of moments (b fitted)	11026.647	11039.290
Method of moments (b from ML)	10913.673	10926.317

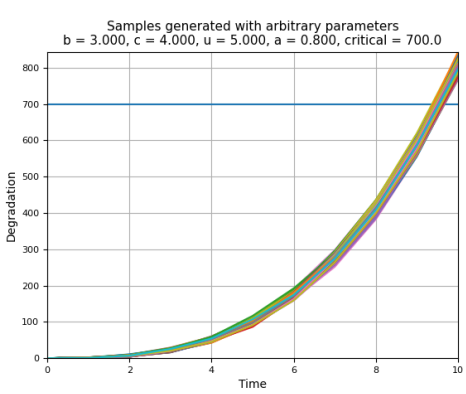
Unlike previous cases, with this set of arbitrary parameters method of maximum likelihood has the best performance.

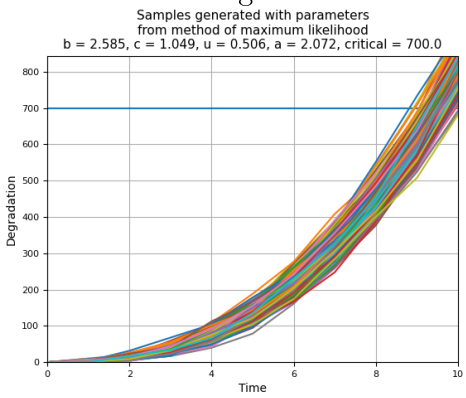
However, when b is greater than 2 the results are different, as shown in the following example:

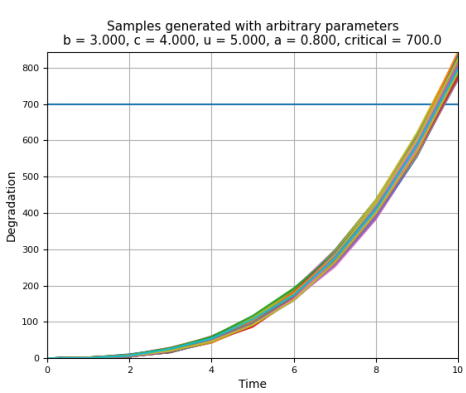
Third set of arbitrary parameters

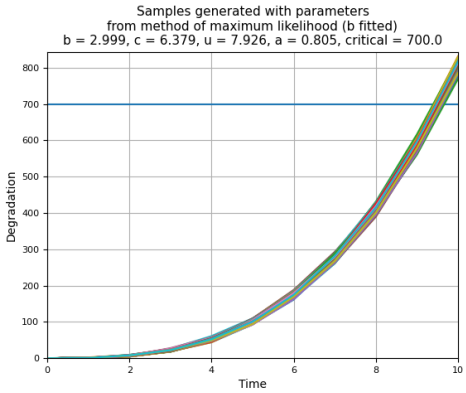
Parameter	Value	Mean	Std	$p_{0.025}$	$p_{0.975}$
b (fitted)	3	2.999	0.047	2.907	3.092
a (fitted)	0.800	0.806	0.087	0.645	0.993
b (method of maximum likelihood)	3	2.585	0.050	2.488	2.672
c (method of maximum likelihood)	4	1.049	0.128	0.817	1.308
u (method of maximum likelihood)	5	0.506	0.082	0.388	0.660
a (method of maximum likelihood)	0.800	2.072			
c (method of maximum likelihood, b fitted)	4	6.379	3.674	2.282	16.683
u (method of maximum likelihood, b fitted)	5	7.926	4.470	2.895	21.036
a (method of maximum likelihood, b fitted)	0.800	0.805			
c (method of moments, b fitted)	4	7.319	6.205	1.803	26.048
u (method of moments, b fitted)	5	9.086	7.585	2.260	30.459
a (method of moments, b fitted)	0.800	0.806			
c (method of moments, b from ML)	4	1.094	0.225	0.727	1.672
u (method of moments, b from ML)	5	0.529	0.127	0.337	0.811
a (method of moments, b from ML)	0.800	2.068			

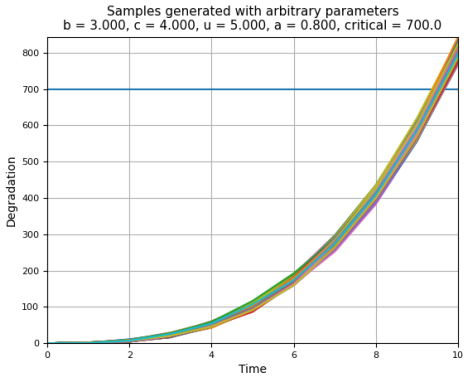
In this case fitting is clearly better, because method of maximum likelihood underestimates b; estimation of a is still good with both method of maximum likelihood and method of moments when the value of b is accurate. As shown in the following graphs, sample deterioration paths are somewhat similar when b is underestimated but the variance is much higher:

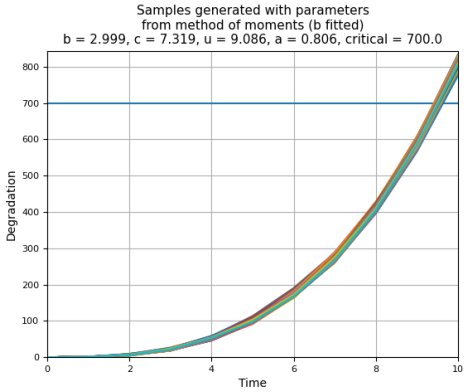


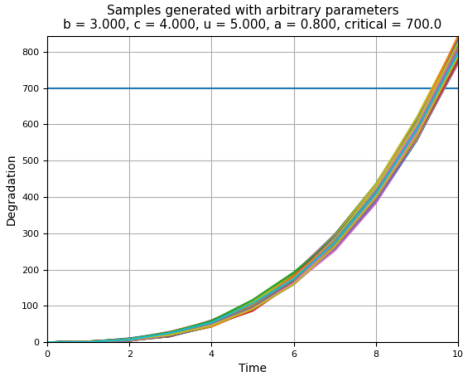


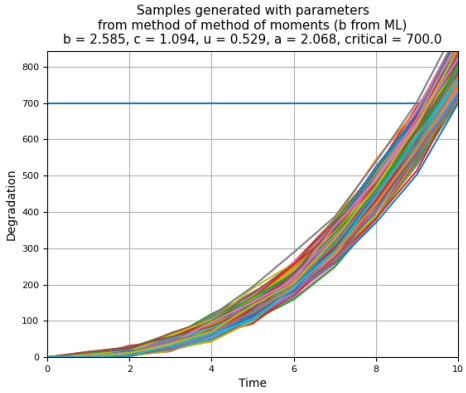






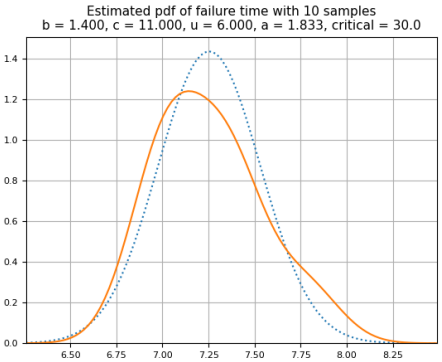


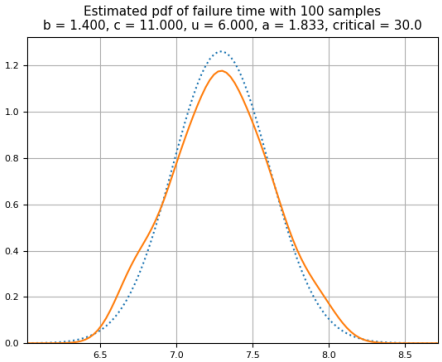


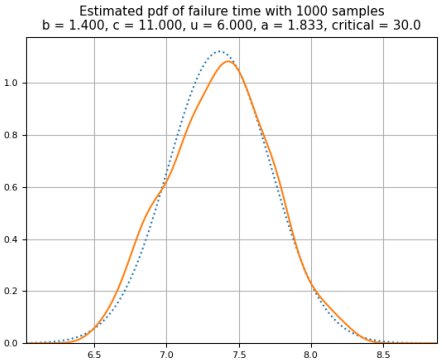


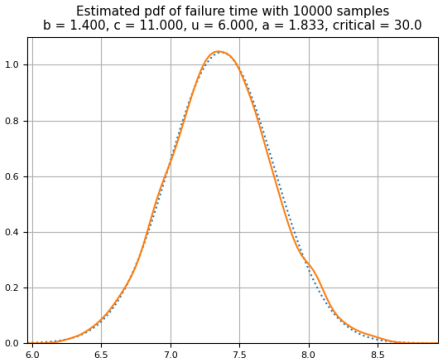
3.2.2 Failure time estimation

The following graphs show the estimated probability density functions obtained with the second set of parameters and increasing number of samples, the dotted line represents a gaussian probability density function with mean and variance equal to sample mean and sample variance:









The two curves become very close as the number of samples increases, which means that the distribution of failure times can be approximated by a normal distribution.

Chapter 4

Application of gamma processes: two case studies

Gamma processes will be applied to two case studies: fatigue-crack growth dataset and GaAs lasers degradation dataset.

4.1 Importing data from dataset and computing parameters

When a dataset is passed as argument, the script reads it and estimates parameters. Dataset should contain $\{t_j\}_j$ and $\{x_{i,j}\}_{i,j}$, where t_j is the j-th inspection time and $x_{i,j}$ is the measured deterioration of component i at time t_j. Since $t_0 = 0$ and $x_{i,0} = 0 \ \forall i$, it is not necessary to insert them in the dataset: the script will automatically add them to t or any x_i that does not have 0 as first value. By default the script assumes that values are separated by comma, but this can be changed with a specific argument.

Dataset can be structured in two ways, and a argument is available to specify which is used:

- Writing $\{t_j\}_j$ in the first line, $\{x_{1,j}\}_j$ in the second line, $\{x_{2,j}\}_j$ in the third line, and so on.
- Writing t_0, $x_{1,0}$, $x_{2,0}$, ..., $x_{n,0}$ in the first line, t_1, $x_{1,1}$, $x_{2,1}$, ..., $x_{n,1}$ in the second line, and so on.

If the user does not specify it, the script assumes that the file uses the first structure. If the argument is not `"rows"` or `"columns"`, an error is raised and the execution stops; same happens if the file is not found.

```
def ReadDataset(datafile, sep, mode):
    if not os.path.isfile(datafile):
        sys.exit('Error: file "{}" not found!'.format(
                                        datafile))
    # 'rows' mode: row 1 contains values of t, other rows
                                    values of x (one object
                                    per row)
    if mode == 'rows':
        with open(datafile) as file:
            lines = file.read().splitlines()
        t = [float(tt) for tt in lines[0].strip().split(sep)]
        x = [[float(xx) for xx in lines[j].strip().split(sep)
                                        ] for j in range(1,
                                        len(lines))]
    # 'columns' mode: row j contains t[j], x1[j], x2[j], ...,
                                     xn[j]
    elif mode == 'columns':
        t = []
        x = []
        with open(datafile) as file:
            for line in file:
                values = line.strip().split(sep)
                t.append(float(values[0]))
                for j in range(1, len(values)):
                    if len(x) < j:
                        x.append([float(values[j])])
                    else:
                        x[j - 1].append(float(values[j]))
    else:
        sys.exit('Error: mode "{}" not recognized!'.format(
                                        mode))
    [...]
```

Once the values are read from dataset, leading zeros are automatically added if necessary, to make sure that $t_0 = 0$ and $x_0 = 0$ for each sample.

```
def ReadDataset(datafile, sep, mode):
    [...]
    # add leading zeros if necessary
    if t[0] > 0:
        t = [0] + t
    xx = []
    for xj in x:
        if xj[0] > 0:
            xx.append([0] + xj)
        else:
            xx.append(xj)
    return t, xx
```

After this, parameters are evaluated using methods described in previous chapters.

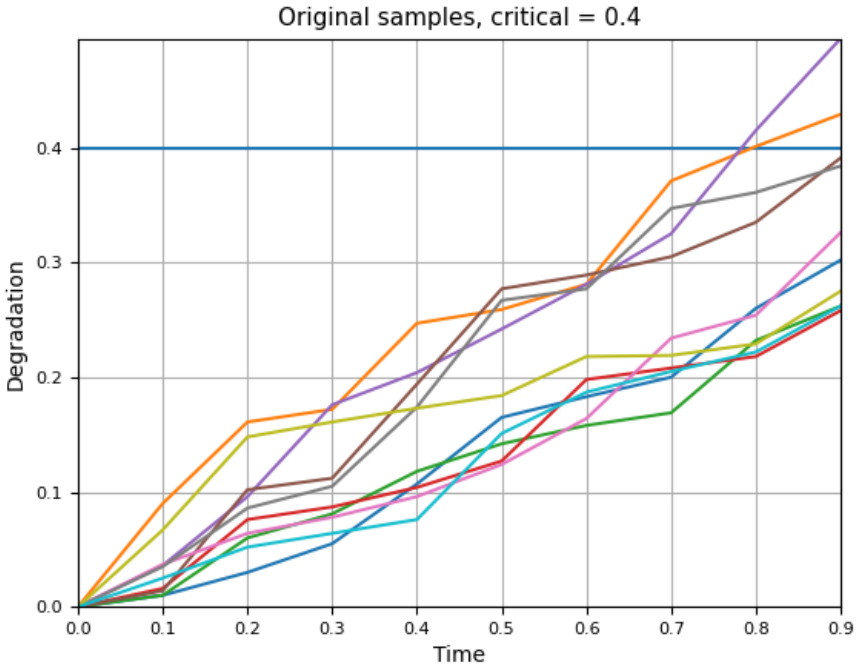
4.2 Fatigue-crack

4.2.1 Overview

This dataset was presented by Rodriguez-Picón et al [16]: it studies the propagation of a crack in a terminal of an electronic device. The function of the terminal is to transfer a signal to a receptor, and the propagation of the crack to a certain critical length can lead to failure of the device given the inability of transferring the signal. The crack growth was measured every 0.1 hundred thousands cycles until 0.9 hundred thousands cycles, for 10 samples: if the length of a crack exceeds the critical level of 0.4 mm the device is considered to have failed.

0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
0.010	0.030	0.055	0.107	0.165	0.183	0.200	0.260	0.302
0.090	0.161	0.172	0.247	0.259	0.281	0.371	0.401	0.429
0.010	0.060	0.081	0.118	0.142	0.158	0.169	0.232	0.262
0.016	0.076	0.087	0.104	0.127	0.198	0.208	0.218	0.258
0.036	0.096	0.176	0.204	0.242	0.281	0.325	0.415	0.495
0.014	0.102	0.112	0.194	0.277	0.289	0.305	0.335	0.391
0.037	0.064	0.078	0.096	0.124	0.164	0.234	0.254	0.326
0.035	0.086	0.105	0.174	0.267	0.277	0.347	0.361	0.384
0.067	0.148	0.161	0.173	0.184	0.218	0.219	0.229	0.275
0.025	0.052	0.064	0.076	0.151	0.187	0.205	0.222	0.262

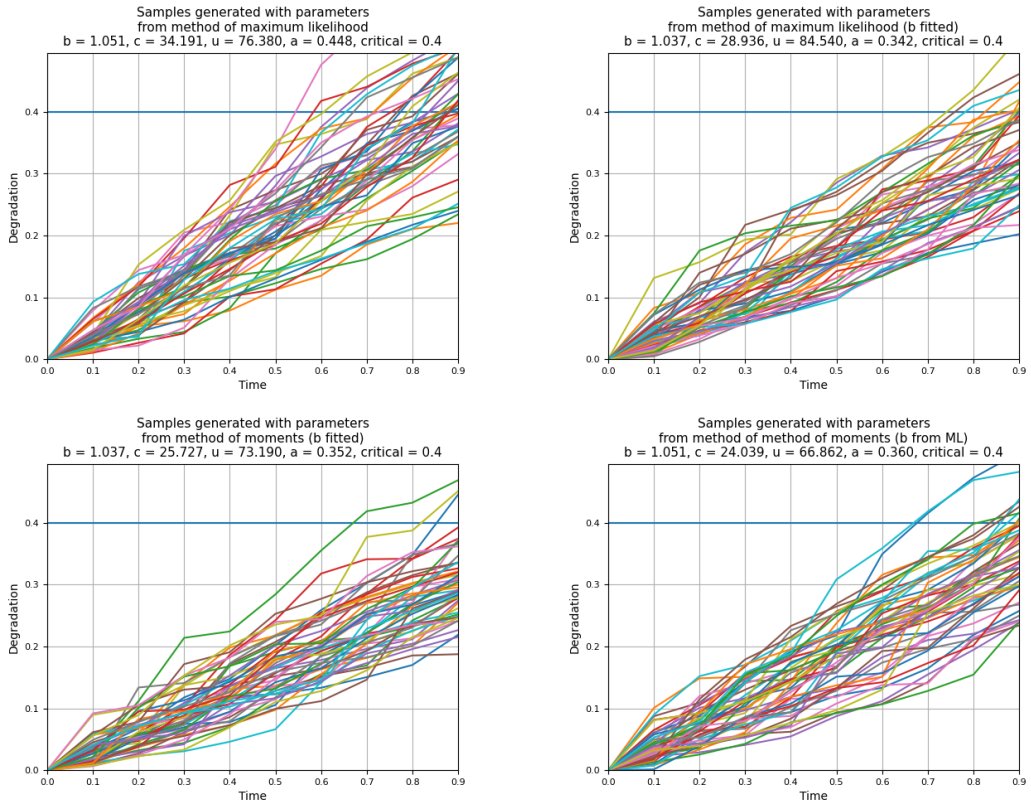
The following graph shows a visual representation of the deterioration level for all samples; two of them exceed the critical level in considered timespan.



4.2.2 Results

Method of maximum likelihood seems to be less accurate than others, both quantitatively and qualitatively: in fact, estimated value of a is significantly different from other methods, standard deviation is much higher, and the graph with randomly generated samples is different. Other methods, including method of maximum likelihood with fitted value of b, return similar results: method of moments is arguably better, because the standard deviation is slightly lower, but the difference is small.

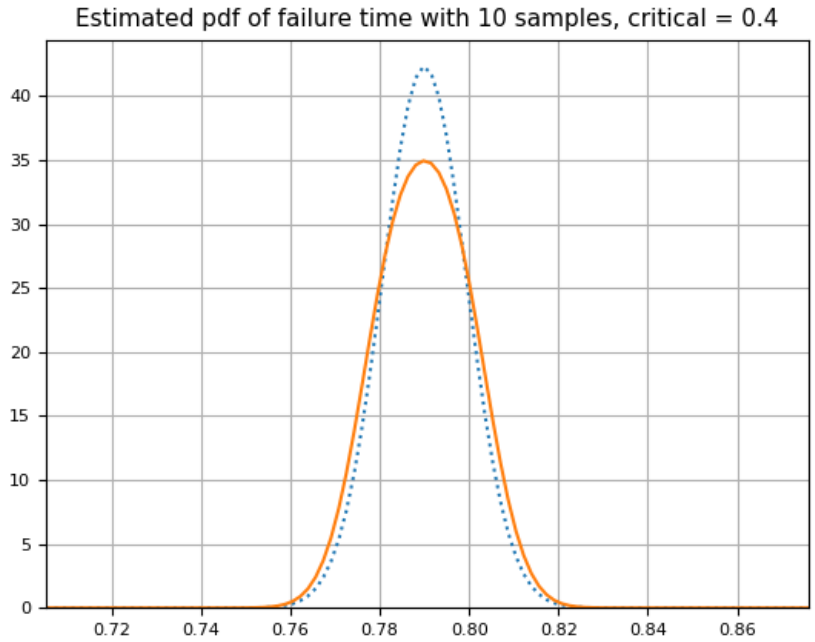
Parameter	Mean	Variance	$p_{0.025}$	$p_{0.975}$
b (fitted)	1.037	0.171	0.776	1.300
a (fitted)	0.367	0.075	0.284	0.460
b (method of maximum likelihood)	1.051	0.188	0.736	1.305
c (method of maximum likelihood)	34.191	18.725	17.462	68.482
u (method of maximum likelihood)	76.380	33.480	40.804	126.692
a (method of maximum likelihood)	0.448			
c (method of maximum likelihood, b fitted)	28.936	10.467	17.177	45.840
u (method of maximum likelihood, b fitted)	84.540	30.724	50.447	125.246
a (method of maximum likelihood, b fitted)	0.342			
c (method of moments, b fitted)	25.727	9.156	15.575	40.671
u (method of moments, b fitted)	73.190	24.575	49.204	109.962
a (method of moments, b fitted)	0.352			
c (method of moments, b from ML)	24.039	9.285	15.678	40.269
u (method of moments, b from ML)	66.862	25.518	39.881	111.401
a (method of moments, b from ML)	0.360			



Both Akaike information criterion and Bayesian information criterion confirm the previous impression: method of moments is less accurate than others and method of moments is slightly better.

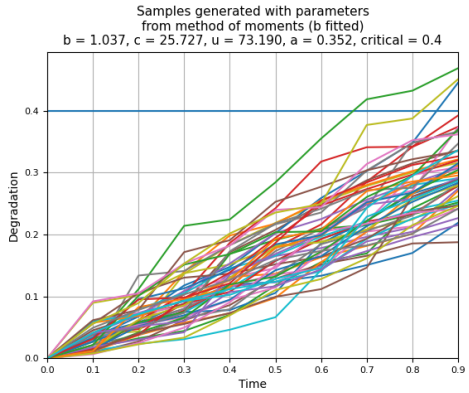
Method	AIC	BIC
Method of maximum likelihood	-399.475	-398.567
Method of maximum likelihood (b fitted)	-415.063	-414.155
Method of moments (b fitted)	-420.732	-419.824
Method of moments (b from ML)	-422.782	-421.874

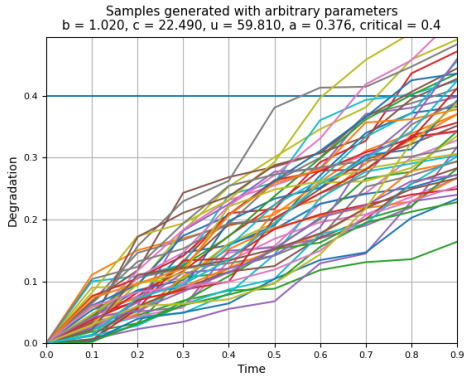
Speaking about time of failure, estimated pdf is shown in the following image. Unfortunately the dataset contains few samples, but the similarity with gaussian distribution can be seen in the graph.



4.2.3 Comparison

The following graphs show a comparison between random samples generated with parameters from method of moments and parameters obtained by Rodriguez-Picón with OpenBUGS software. Value of b and a are very similar, but in the latter case the variance is higher because of different values of c and u, and the effect is evident in the graph.





However, AIC is very close with a value of -430 in the best case: this means that these two methods perform similarly.

4.3 GaAs lasers

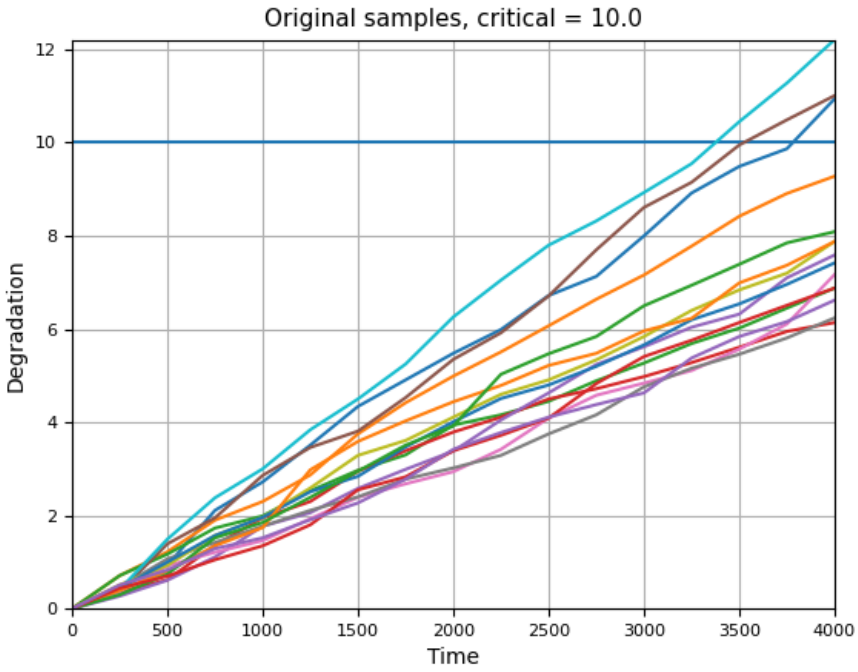
4.3.1 Overview

This dataset was presented by Meeker and Escobar [15]: it studies the effects of degradation on some laser devices. They have a decrease in light output when degradation increases, but some of them mantain the output constant by increasing the operating current; when it gets too high the device is considered to have failed. In this case the GaAs lasers have been studied at 80°, which is much higher than the use temperature, to accelerate failures and obtain data more quickly. 15 samples have been observed to measure the percent increase in operating current at inspection times $t_j = 250, 500, 750, ..., 4000$ hours.

250	500	750	1000	1250	1500	1750	2000	2250	2500	2750	3000	3250	3500	3750	4000
0.47	0.93	2.11	2.72	3.51	4.34	4.91	5.48	5.99	6.72	7.13	8.00	8.92	9.49	9.87	10.94
0.71	1.22	1.90	2.30	2.87	3.75	4.42	4.99	5.51	6.07	6.64	7.16	7.78	8.42	8.91	9.28
0.71	1.17	1.73	1.99	2.53	2.97	3.30	3.94	4.16	4.45	4.89	5.27	5.69	6.02	6.45	6.88
0.36	0.62	1.36	1.95	2.30	2.95	3.39	3.79	4.11	4.50	4.72	4.98	5.28	5.61	5.95	6.14
0.27	0.61	1.11	1.77	2.06	2.58	2.99	3.38	4.05	4.63	5.24	5.62	6.04	6.32	7.10	7.59
0.36	1.39	1.95	2.86	3.46	3.81	4.53	5.35	5.92	6.71	7.70	8.61	9.15	9.95	10.49	11.01
0.36	0.92	1.21	1.46	1.93	2.39	2.68	2.94	3.42	4.09	4.58	4.84	5.11	5.57	6.11	7.17
0.46	1.07	1.42	1.77	2.11	2.40	2.78	3.02	3.29	3.75	4.16	4.76	5.16	5.46	5.81	6.24
0.51	0.93	1.57	1.96	2.59	3.29	3.61	4.11	4.60	4.91	5.34	5.84	6.40	6.84	7.20	7.88
0.41	1.49	2.38	3.00	3.84	4.50	5.25	6.26	7.05	7.80	8.32	8.93	9.55	10.45	11.28	12.21
0.44	1.00	1.57	1.96	2.51	2.84	3.47	4.01	4.51	4.80	5.20	5.66	6.20	6.54	6.96	7.42
0.39	0.80	1.35	1.74	2.98	3.59	4.03	4.44	4.79	5.22	5.48	5.96	6.23	6.99	7.37	7.88
0.30	0.74	1.52	1.85	2.39	2.95	3.51	3.92	5.03	5.47	5.84	6.50	6.94	7.39	7.85	8.09
0.44	0.70	1.05	1.35	1.80	2.55	2.83	3.39	3.72	4.09	4.83	5.41	5.76	6.14	6.51	6.88
0.51	0.83	1.29	1.52	1.91	2.27	2.78	3.42	3.78	4.11	4.38	4.63	5.38	5.84	6.16	6.62

The following graph shows a visual representation of the deterioration

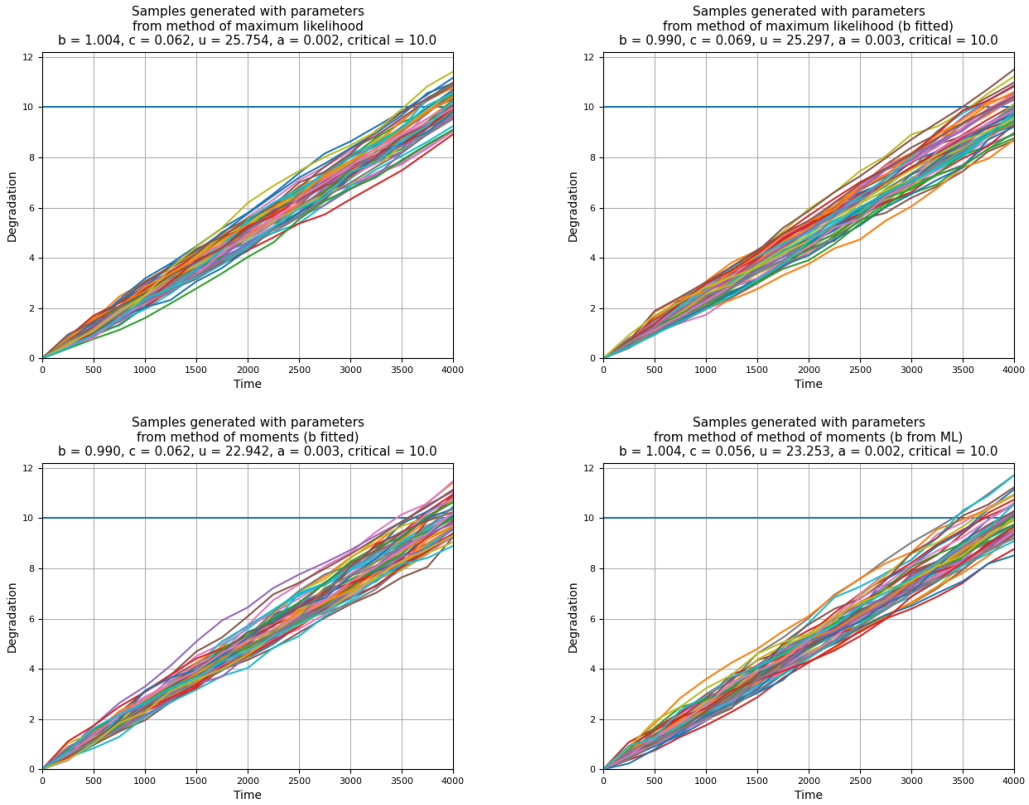
level for all samples; three of them exceed the critical level in considered timespan.



4.3.2 Results

With this dataset performance is similar with all methods, both qualitatively and quantitatively: unlike the previous case, method of maximum likelihood performs well. Method of moments with b estimated from maximum likelihood method seems to be better, because its samples span over a higher interval which is closer to the one of the original samples, but again the difference is small.

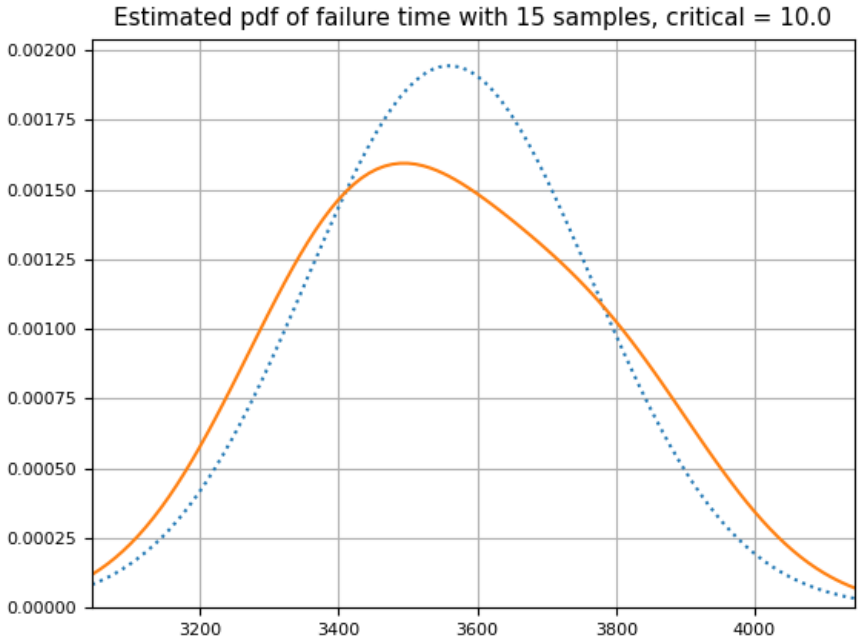
Parameter	Mean	Variance	$p_{0.025}$	$p_{0.975}$
b (fitted)	0.990	0.072	0.870	1.098
a (fitted)	0.003	0.001	0.001	0.005
b (method of maximum likelihood)	1.004	0.060	0.917	1.088
c (method of maximum likelihood)	0.062	0.050	0.019	0.161
u (method of maximum likelihood)	25.754	10.528	13.925	44.481
a (method of maximum likelihood)	0.002			
c (method of maximum likelihood, b fitted)	0.069	0.049	0.018	0.152
u (method of maximum likelihood, b fitted)	25.297	10.510	13.659	44.018
a (method of maximum likelihood, b fitted)	0.003			
c (method of moments, b fitted)	0.062	0.047	0.015	0.147
u (method of moments, b fitted)	22.942	10.392	11.292	40.165
a (method of moments, b fitted)	0.003			
c (method of moments, b from ML)	0.056	0.048	0.017	0.152
u (method of moments, b from ML)	23.253	10.448	11.299	40.544
a (method of moments, b from ML)	0.002			



However, both Akaike information criterion and Bayesian information criterion suggest that method of moments performs drastically better than method of maximum likelihood, with b fitted as the best technique.

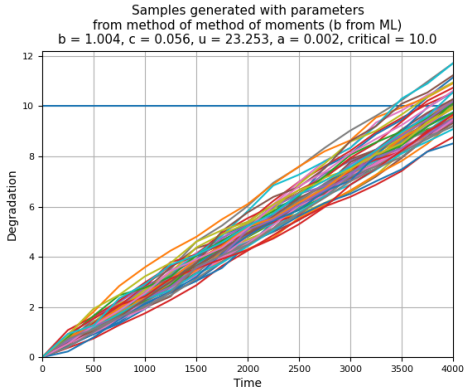
Method	AIC	BIC
Method of maximum likelihood	126.625	128.749
Method of maximum likelihood (b fitted)	126.461	128.585
Method of moments (b fitted)	75.313	77.438
Method of moments (b from ML)	82.265	84.389

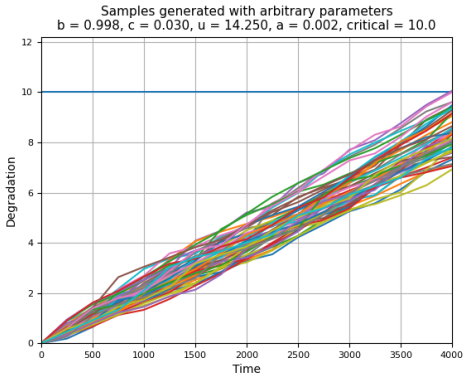
Speaking about time of failure, similar considerations as before hold: athough the dataset is small, similarity with gaussian distribution can be seen in the graph.



4.3.3 Comparison

In this case samples generated with Rodriguez-Picón's parameters changes drastically: variance is similar, but resulting deterioration paths are significantly lower, and almost no sample reaches failure (only 2 or 3 on average, out of 50 samples). Unfortunately his article does not provide AIC or BIC for this case study.


Samples generated with parameters
from method of method of moments (b from ML)
b = 1.004, c = 0.056, u = 23.253, a = 0.002, critical = 10.0
Degradation
Time
0
2
4
6
8
10
12
0
500
1000
1500
2000
2500
3000
3500
4000


Samples generated with arbitrary parameters
b = 0.998, c = 0.030, u = 14.250, a = 0.002, critical = 10.0
Degradation
Time
0
2
4
6
8
10
12
0
500
1000
1500
2000
2500
3000
3500
4000

Chapter 5

Conclusions

Gamma processes represent a powerful method to study the deterioration of objects, which is very useful because it is a key problem in maintenance to lower the costs and they have a lot of possibile applications. Of course there are difficulties: each scenario requires specific adaptation of the methods, because we have to find a way to measure deterioration and gather data from various samples, which is not always easy to do. However, results are encouraging: examples with arbitrary parameters show that methods perform well, and analyzed case studies show that they can be applied to real-life scenarios with good results. This is encouraging for future applications but also to study and develop more complex models. For example one can study the case when critical level is not deterministic but stochastic with some known distribution (e.g. uniform or normal); or there could be more components that deteriorate together, leading to a multivariate case.

Chapter 6

Appendix

Recall that the likelihood function of the observed deterioration increments is

$$\ell(\delta_1, ..., \delta_n | c, u) = \prod_{j=1}^{n} \frac{u^{cw_j}}{\Gamma(cw_j)} \delta_j^{cw_j - 1} e^{-u\delta_j}$$

where $\delta_j = x_j - x_{j-1}$ and $w_j = t_j^b - t_{j-1}^b$, so the loglikelihood is

$$\ell\ell(\delta_1, ..., \delta_n | c, u) = \log \prod_{j} \frac{u^{cw_j}}{\Gamma(cw_j)} \delta_j^{cw_j - 1} e^{-u\delta_j}$$

$$= \sum_{j} \log \left(\frac{u^{cw_j}}{\Gamma(cw_j)} \delta_j^{cw_j - 1} e^{-u\delta_j} \right)$$

Before calculating the first partial derivatives of $\ell\ell$ with respect to c and u, it is useful to note that given $\alpha \in \mathbb{R}$ and $f : \mathbb{R} \to \mathbb{R}$ then

$$\frac{d}{dx} \log(\alpha f(x)) = \frac{d}{dx} (\cancel{\log \alpha} + \log f(x)) = \frac{d}{dx} \log f(x)$$

This means that some terms can be removed from logarithm argument when calculating partial derivatives.

First partial derivative with respect to c is

$$\frac{\partial}{\partial c} \ell\ell(\delta_1, ..., \delta_n | c, u) = \frac{\partial}{\partial c} \sum_{j} \log \left(\frac{u^{cw_j}}{\Gamma(cw_j)} \delta_j^{cw_j - 1} e^{-u\delta_j} \right)$$

$$= \sum_{j} \frac{\partial}{\partial c} \log \left(\frac{u^{cw_j}}{\Gamma(cw_j)} \delta_j^{cw_j - 1} \cancel{e^{-u\delta_j}} \right)$$

$$= \sum_j \frac{\Gamma(cw_j)}{u^{cw_j}\delta_j^{cw_j-1}} \frac{\partial}{\partial c}\left(\frac{u^{cw_j}\delta_j^{cw_j-1}}{\Gamma(cw_j)}\right)$$

$$= \sum_j \frac{\Gamma(cw_j)}{u^{cw_j}\delta_j^{cw_j-1}} \left(\frac{1}{\Gamma^2(cw_j)}\left(\frac{\partial}{\partial c}(u^{cw_j}\delta_j^{cw_j-1})\Gamma(cw_j) - u^{cw_j}\delta_j^{cw_j-1}\frac{\partial\Gamma(cw_j)}{\partial c}\right)\right)$$

$$= \sum_j \frac{1}{u^{cw_j}\delta_j^{cw_j-1}\Gamma(cw_j)}\left(\frac{\partial}{\partial c}(u^{cw_j}\delta_j^{cw_j-1})\Gamma(cw_j) - u^{cw_j}\delta_j^{cw_j-1}w_j\Gamma'(cw_j)\right)$$

$$= \sum_j \frac{1}{u^{cw_j}\delta_j^{cw_j-1}}\left(w_j\log(u)u^{cw_j}\delta_j^{cw_j-1} + u^{cw_j}w_j\log(\delta_j)\delta_j^{cw_j-1} - u^{cw_j}\delta_j^{cw_j-1}w_j\frac{\Gamma'(cw_j)}{\Gamma(cw_j)}\right)$$

$$= \sum_j \frac{w_j \cancel{u^{cw_j}}\cancel{\delta_j^{cw_j-1}}}{\cancel{u^{cw_j}}\cancel{\delta_j^{cw_j-1}}}(\log u + \log\delta_j - \psi(cw_j))$$

$$= \log u\sum_j w_j + \sum_j w_j(\log\delta_j - \psi(cw_j))$$

$$= t_n^b\log u + \sum_j w_j(\log\delta_j - \psi(cw_j))$$

First partial derivative with respect to u is

$$\frac{\partial}{\partial u}\ell\ell(\delta_1,...,\delta_n|c,u) = \sum_j \frac{\partial}{\partial u}\log\left(\frac{u^{cw_j}}{\cancel{\Gamma(cw_j)}}\cancel{\delta_j^{cw_j-1}}e^{-u\delta_j}\right)$$

$$= \sum_j \frac{1}{u^{cw_j}e^{-u\delta_j}}\frac{\partial}{\partial u}u^{cw_j}e^{-u\delta_j}$$

$$= \sum_j \frac{1}{u^{cw_j}e^{-u\delta_j}}(cw_ju^{cw_j-1}e^{-u\delta_j} + u^{cw_j}(-\delta_je^{-u\delta_j}))$$

$$= \sum_j \frac{1}{\cancel{u^{cw_j}}\cancel{e^{-u\delta_j}}}\cancel{u^{cw_j}}\cancel{e^{-u\delta_j}}(cw_ju^{-1} - \delta_j)$$

$$= \frac{c}{u}\sum_j w_j - \sum_j \delta_j = \frac{c}{u}t_n^b - x_n$$

To maximize the likelihood function of the observed deterioration increments these two partial derivatives must be equal to zero, i.e.

$$\begin{cases}\dfrac{\partial}{\partial c}\ell\ell(\delta_1,...,\delta_n|c,u) = 0\\ \dfrac{\partial}{\partial u}\ell\ell(\delta_1,...,\delta_n|c,u) = 0\end{cases}$$

$$\begin{cases} t_n^b \log u + \sum_j w_j(\log \delta_j - \psi(cw_j)) = 0 \\ \frac{c}{u} t_n^b - x_n = 0 \end{cases}$$

$$\begin{cases} \sum_j w_j(\psi(cw_j) - \log \delta_j) = t_n^b \log\left(\frac{c}{u} t_n^b\right) \\ u = \frac{c}{u} t_n^b \end{cases}$$

This is the system (2.5).

Bibliography

[1] J.M. van Noortwijk (2007), A survey of the application of gamma processes in maintenance, *Reliability Engineering and System Safety* (vol. 94, p. 2-21)

[2] R. Dekker (1996), Applications of maintenance optimization models: a review and analysis, *Reliability Engineering and System Safety* (vol. 51, p. 229-240)

[3] M.D. Pandey, X.-X. Yuan, J.M. van Noortwijk (2007), A comparison of probabilistic deterioration models for life-cycle management of structures, *Advances in Engineering Structures, Mechanics & Construction* (vol. 140, p. 735-746)

[4] R.E. Barlow, F. Proschan, *Mathematical theory of reliability* (1965), Society for Industrial and Applied Mathematics

[5] A.P. Basu, N. Ebrahimi (1983), On the reliability of stochastic systems, *Statistics & Probability Letters* (vol. 1, p. 265-267)

[6] A. Mercer, C.S. Smith (1959), A random walk in which the steps occur randomly in time, *Biometrika* (vol. 46, p. 30-35)

[7] B.R. Ellingwood, Y. Mori (1993), Probabilistic methods for condition assessment and life prediction of concrete structures in nuclear power plants, *Nuclear Engineering and Design* (vol. 142, p. 155-166)

[8] E. Çinlar, Z.P. Bazant, E. Osman (1977), Stochastic process for extrapolating concrete creep, *ASCE J Eng Mech Div* (vol. 103, p. 1069-1088)

[9] R.P. Nicolai, R. Dekker, J.M. van Noortwijk (2007), A comparison of models for measurable deterioration: An application to coatings on steel structures, *Reliability Engineering & System Safety* (vol. 92, p. 1635-1650)

[10] Python: `https://www.python.org/`

[11] NumPy: `https://numpy.org/`

[12] SciPy: `https://www.scipy.org/`

[13] Matplotlib: `https://matplotlib.org/`

[14] E. Wit, E. van den Heuvel, J. Romeijn (2012), 'All models are wrong...': an introduction to model uncertainty, *Statistica Neerlandica* (vol. 66, p. 217-236)

[15] W.Q. Meeker, L.A. Escobar, *Statistical Methods for Reliability Data* (1998), Wiley-Interscience

[16] L.A. Rodríguez-Picón, A.P. Rodríguez-Picón, L.C. Méndez-González, M.I. Rodríguez-Borbón, A. Alvarado-Iniesta (2017), Degradation modeling based on gamma process models with random effects, *Communications in Statistics - Simulation and Computation* (vol. 47:6, p. 1796-1810)