A model of aging as accumulated damage matches observed mortality patterns and predicts the life-extending effects of prospective interventions

Chris Phoenix · Aubrey D. N. J. de Grey

Received: 14 March 2007 / Accepted: 24 August 2007 / Published online: 18 September 2007
© American Aging Association 2007

Abstract The relative insensitivity of lifespan to environmental factors constitutes compelling evidence that the physiological decline associated with aging derives primarily from the accumulation of intrinsic molecular and cellular side-effects of metabolism. Here we model that accumulation starting from a biologically based interpretation of the way in which those side-effects interact. We first validate this model by showing that it very accurately reproduces the distribution of ages at death seen in typical populations that are well protected from age-independent causes of death. We then exploit the mechanistic basis of this model to explore the impact on lifespans of interventions that combat aging, with an emphasis on interventions that repair (rather than merely retard) the direct molecular or cellular consequences of metabolism and thus prevent them from accumulating to pathogenic levels. Our results strengthen the case that an indefinite extension of healthy and total life expect-

tancy can be achieved by a plausible rate of progress in the development of such therapies, once a threshold level of efficacy of those therapies has been reached.

Keywords Aging · Damage · Intervention · Lifespan · Longevity escape velocity · Rejuvenation · Reserve · Simulation

Introduction

The actuary Benjamin Gompertz first observed more than 180 years ago that, in the British population he studied, the proportion of people reaching age N who died before reaching age $N+1$ was approximated with remarkable precision by an exponential function of N across the range of ages at which most people died (Gompertz 1825). This relationship has since been found to hold, with great accuracy, across all human and animal populations that (1) exhibit aging at all and (2) do not suffer appreciable mortality from causes unrelated to aging. What is its biological basis?

Gavrilov and Gavrilova (1991, 2004) have explored in some detail an abstract model of aging that is intended to shed light on this question. They have focussed on the concept of the body as a machine that is built with an initially high level of functional redundancy. Aging is then conceptualised as the progressive failure of components of this machine, which increasingly diminishes its degree of redundancy, such that eventually individual components become essential, and the machine fails (the

C. Phoenix
Center for Responsible Nanotechnology,
213 Eastern Parkway, Suite 11,
Brooklyn, NY 11238, USA

A. D. N. J. de Grey (✉)
Methuselah Foundation,
P.O. Box 1143, Lorton, VA 22079, USA
e-mail: aubrey@sens.org

organism dies) when a component that has become essential fails. Analyses using reliability theory have shown that their model can approximate reasonably accurately the mortality patterns in real populations. While this model of aging possesses some of the characteristics of real-life organisms, it is unrealistic in the important sense that it depicts the organism as made of components that are (1) individually non-aging and (2) not replaced over time. Depending on the scale at which an organism is described, one or other of these properties is true, but at no scale are they both true: atoms do not age, but they are constantly replaced within all cells and tissues, whereas higher levels of complexity (macromolecules, cells, organs) are sometimes not replaced but, if not replaced, do age.

A related body of work concerns the relationship between functional decline and mortality in human populations, such as the average number of years that people survive after becoming incapable of a certain number of standard “activities of daily living” (ADLs) (e.g. Fries 1980; Manton et al. 2006; Rockwood and Mitnitski 2006; Ukraintseva and Yashin 2001). While this type of study can provide useful information about, for example, the extent to which functional deficits exacerbate each other, it is necessarily limited to analysis of the later stages of life, when functional decline is present; it cannot illuminate the *lifelong* accumulation of initially latent molecular and cellular changes that is the necessary precursor of that decline.

In developing policy concerning investment in the development of interventions to combat aging, it is vital that society has maximally reliable predictors of the impact of a given intervention (or sequence of interventions) on healthy and total lifespans. This requires an exploration of the relationship between lifelong biological aging and mortality in a more biologically faithful and comprehensive fashion than has been achieved in the work outlined above.

In order to do this, we have developed a model based directly on the representation of aging as accumulation of initially latent damage. Metabolism (here defined in its broad sense, as the entire set of biochemical and cellular processes that maintain organismal function) is a highly robust homeostatic network that maintains the organism in a functional state in the face of phenomena that transiently perturb that state—infections, meals, exertion, etc. The component processes that make up metabolism have side-effects, however, some of which accumulate over

the life course. These include the literal accumulation of material at the molecular (e.g. proteinaceous aggregates in the brain) or cellular (e.g. anergic T cells) level, structural changes to long-lived molecules (e.g. stiffening of extracellular collagen, mutations in the DNA of long-lived cells), and loss of functional components (e.g. the death and non-replacement of cells in the heart and in some parts of the brain) (de Grey et al. 2002; de Grey 2005). These progressive changes can appropriately be grouped under the single term “damage”, because they antagonise the efforts of our metabolic network to keep the body fully functional. However, only the last (loss of cells) truly fits the “loss of redundancy” characterisation of aging.

The effect of this accumulating damage on health and lifespan is not simple. First, each type of damage affects different aspects of metabolism to different degrees, such that in principle a single type could kill the organism once it rose to a sufficient level even if all other types of damage were totally absent. Second, and conversely, types of damage can be considered in clusters, in which the members of a cluster contribute additively to impair essentially the same set of aspects of metabolism by the same mechanisms: for example, the many chemically distinct types of protein–protein crosslink that accumulate in long-lived structures such as the artery wall all contribute to hypertension by making the wall less elastic. Thirdly, metabolism includes many processes that limit the rate of accumulation of damage; thus, that rate will tend to accelerate as the overall level of damage (of all types) increases and makes metabolism’s homeostatic task harder, even before it begins to fail in that task (i.e. functional decline emerges). Fourth, death is not a deterministic outcome of the progression to a given level of damage, because death generally results from the body’s inability to cope with a stochastic, often external, challenge such as an infection or an injury: thus, one’s level of damage dictates one’s *risk* of death in the near future rather than causing instant death. The model presented here incorporates all these features.

A biologically realistic model can potentially be used to predict the degree to which morbidity and mortality might be postponed by various types of medical intervention. Hitherto, work in this area has explored only the effects of curing age-related diseases, and has been based on the proportions of people who die of those diseases at various ages. The main limitation of such an approach is that it cannot explore the processes leading

up to death from each cause, only the end result. Thus, it is unsuited to the modelling of therapies that are applied relatively early in the development of age-related pathology. Therapies of this sort are the key target of the biomedical gerontologist, because they potentially slow the progression of all age-related pathologies, on account of the accelerating and causally interconnected damage accumulation mentioned above.

In this work we examine interventions that each eliminate a single type of damage. Specifically, we restrict ourselves to therapies that remove the damage itself, rather than ones that prevent it from being laid down in the first place. Therapies that remove damage are of particular interest for two main reasons: firstly they have the potential to diminish the total damage present in the organism, i.e. to restore increased youthfulness, and secondly they may be more amenable to combined application to the same individual without unanticipated interactions, since they target the initially inert side-effects of metabolism, rather than bioactive components of metabolism itself. Hereafter we refer to such therapies as “cures” for components of aging.

A third difference between retardation and repair is illustrated in Fig. 1. Real-world interventions in a process as complex as aging will not be totally comprehensive in the foreseeable future, if ever; thus, the question is which type of imperfect intervention will be of the greatest benefit. A halving (blue line) of the default rate of aging (red line), applied to people already in middle age, will benefit them much less than a periodically applied therapy that removes half the damage (pink line). But this difference becomes more dramatic when improvements with time in the efficacy of the therapy are taken into account. The second and subsequent applications of the non-improving repair therapy in Fig. 1 (pink line) are progressively less effective as the total damage becomes more and more dominated by the life-long accumulation of the types of damage that the therapy does not affect. But if, by the second time the repair therapy is applied, it has been improved to the point where it can fix half the damage that it originally could not fix, and the third time it can fix half the rest, and so on, the result (Fig. 1, orange line) is an indefinite avoidance of a pathogenic level of damage, even though the therapy is never perfect. This can only be achieved with a retardation therapy if the improvement in its efficacy is far more rapid or begun at a much earlier age—and, of course, the beneficiaries of such a therapy grow steadily closer to frailty all the time (Fig. 1, green line).

Fig. 1 Effects of various anti-aging therapeutic scenarios on a typical individual who is in middle age when the therapies are initiated. The red line indicates the default trajectory of accumulation of damage (or, equivalently, loss of reserve) in the absence of therapies; it should be stressed that this is for illustration only and that we make no assumption that the rate of this process is linear with age. It is presumed that there is a “frailty threshold” of loss of reserve, beyond which intervention becomes much harder as pathologies are emerging and exacerbating their own rate of progress. Blue line A therapy that halves the accumulation of damage (until it reaches the frailty threshold) but leaves pre-existing damage unaltered. Green line A therapy that initially halves the accumulation of damage and whose efficacy is doubled every 7 years, but which always leaves pre-existing damage unaltered. Pink line A therapy that removes half the damage. Orange line A therapy that initially removes half the damage, but 20 years later has been improved so that it removes half the damage that it previously could not, and undergoes the same improvement every 20 years thereafter. Note: colours appear in online version of this article and in reprints available on request

Methods

A program was written to simulate the accumulation of damage due to aging processes. The aggregate simulated deaths produce a simulated survival curve, which is fitted to match observed United States mortality data from 1999 (Wilmoth 1997). Once the fitted parameters are determined, the impact on survival of a given schedule of removal of parts of the damage is calculated.

Each simulated person has multiple categories of aging process. Each category has multiple mechanisms, and each mechanism contributes additively to the damage level of its category. Each mechanism has two contributors to its rate: a basal component that adds a constant amount of damage each year, and a feedback component that increases damage in proportion to the person’s existing damage (the average of its category damages), creating an accelerating increase in damage.

The simulation parameters are in three categories: those that define the simulation (“structural” parameters, such as the number of categories and mechanisms of damage and the relative weights of damage mechanisms), those that are adjusted to match the data provided (“fitting” parameters), and those that define

the cure schedule (“cure” parameters). Structural parameters can be varied to explore the implications of various assumptions about the aging process.

The program is used in two stages. First, for a given set of structural parameters, the fitting parameters are adjusted to maximise the match to observed mortality data. Then, with optimised fitting parameters determined, the effects of curing various damage mechanisms on various schedules can be simulated. Remember that each category of damage is composed of multiple mechanisms; each mechanism has an independently set weight. By setting a variety of weights of different damage types and introducing cures on various schedules, it is possible to simulate a wide variety of scenarios.

Many different measures of the goodness-of-fit of the simulated curve to the observed data are possible; however, as set out below, our simulation achieved so high a goodness-of-fit that it was unnecessary to choose between these alternatives (i.e. the optimised values of the fitting parameters were virtually identical whatever measure was chosen).

All internal variables are updated once per year. After each damage update, a probability of death is calculated for each person, based on the maximum damage level in any category. A uniform-distribution random number is compared with the death probability; if the former is lower, the person dies. The varying rate of increase in damage at various ages, combined with the shape of the death probability function, determine the probability of death at various ages.

The fitting parameters are used to set the basal and feedback rates for each mechanism in each category. These parameters are fixed at age 0; in subsequent years, the damage of each type in each person is the result of a deterministic feedback between the person’s various mechanisms.

The two damage rates (basal and feedback) must be assigned for each person for each mechanism in each category. Thus, we define a lognormal distribution shaped by two parameters (mean and standard deviation), and then for each person we assign a basal rate and a feedback rate for each mechanism. However, there will be heterogeneity between individuals for a given mechanism, and also heterogeneity between mechanisms within a given individual. To reflect this, for both basal and feedback rates we also define a pan-mechanism rate for each person, taken from the same lognormal distribution, which is added to each mechanism-specific rate for that person after multiplying by

a weighting factor that is the same for every person. (This weighting factor is hereafter termed the “homogeneity” of the rate.) Thus, each of the two damage rate components (basal and feedback) has three of the six fitting parameters: the mean and standard deviation of the distribution, and the homogeneity selector.

In simulations in which aging mechanisms are cured, curing a mechanism sets both its accumulated damage and damage accumulation rate to zero.

One additional fitting parameter determines the probability of death in the coming year given a particular damage level. The curve is exponential and chosen to have a value of 0 at damage level 0 and 1 at damage level 1; this is designed to reflect the principle that, both intuitively and as seen when measured in terms of types of functional deficit, there is a threshold level of damage that is guaranteed to be fatal within a short time.

Structural parameters

- N_CAT: The number of damage categories each person has
- N_MECH: The number of mechanisms in each category
- MECH_WEIGHT_m (for each m from 1..N_MECH): The contribution of a mechanism to its category

Fitting parameters

- BASAL_M: The mean basal damage rate
- BASAL_SD: The standard deviation of the basal damage rate
- EXP_M: The mean feedback damage rate
- EXP_SD: The standard deviation of the feedback damage rate
- BASAL_HOMO: The homogeneity of basal damage rate in a single person
- EXP_HOMO: The homogeneity of feedback damage rate in a single person
- SHAPE_DEATH: Fitting parameter for probability-of-death curve

Internal variables (for each person)—set at time $t=0$

- PB: Basal rate for all mechanisms: lognorm (BASAL_M, BASAL_SD)
- PE: Feedback rate for all mechanisms: lognorm (EXP_M, EXP_SD)

- $MB_{c,m}$: Basal rate for each mechanism, where $m=1..N_MECH$ and $c=1..N_CAT$
- $\text{lognorm}(BASAL_M, BASAL_SD)*(1-BASAL_HOMO) + PB*BASAL_HOMO$
- $ME_{c,m}$: Feedback rate for each mechanism, where $m=1..N_MECH$ and $c=1..N_CAT$
- $\text{lognorm}(EXP_M, EXP_SD)*(1-EXP_HOMO) + PE*EXP_HOMO$
- $DI_M_{m,c}(t)$: Damage increment for each mechanism at time t
- $DI_M_{m,c}(t) = MB_{c,m} + ME_{c,m} * PD(t-1)$
- $D_M_{m,c}(t)$: Cumulative damage for each mechanism at time t
- $D_M_{m,c}(0) = 0$
- $D_M_{m,c}(t) = DI_M_{m,c}(t) + D_M_{m,c}(t-1)$
- $FATAL(t)$: Fatality challenge each year.
- $FATAL(t) = \text{uniform}(0..1)$
- $D_C_c(t)$: Damage for each category
- $D_C_c(0) = 0$
- $D_C_c(t) = [\text{SUM } m=1 \text{ to } N_MECH] D_M_{m,c}(t)$
- $PD(t)$: Damage for the person
- $PD(t) = [\text{SUM } c=1 \text{ to } N_CAT] D_C_c(t)$
- $DMAX(t)$: Category-maximum damage for the person
- $DMAX(t) = [\text{MAX } c=1 \text{ to } N_CAT] D_C_c(t)$

Deaths and cures

- $rPd(\text{damage}) = 1/(2^{(\text{SHAPE_DEATH}*(1-\text{damage}))})$
- $sPd(\text{damage}) = (rPd(\text{damage}) - rPd(0))/(1 - rPd(0))$

If $sPd(DMAX(t)) > FATAL(t)$, the person dies at age t .

To cure a mechanism, $MB_{c,m}$, $ME_{c,m}$, and $D_M_{c,m}(t)$ are all set to 0.

Results

Fitting human mortality data

Figure 2a and b illustrate our first major finding: our model can generate a mortality pattern that fits contemporary human mortality data extremely well, better even than the Gompertz formula. Figure 2a shows mortality data for the United States population in 1999 (orange), the best-fit Gompertz curve (yellow) and the best-fit results from our simulation (green). The goodness-of-fit of our simulation to the data exceeds $r^2=0.99$. That this is even better than the Gompertz fit is most strikingly illustrated when the data are plotted as distribution of death, as shown in Fig. 2b: it can be clearly seen that

the superiority does not derive simply from a better fit at early ages, but at the ages at which most deaths occur.

Another reassuring measure of the biological realism of our simulation is the distribution of average damage level when plotted against age. As shown in Fig. 3, simulated damage accumulates during old age at roughly twice the rate at which it accumulates during youth. The inter-individual variation in rate of simulated aging is also plausible, with the slowest-aging simulated individuals having a damage level at age 85 that the fastest agers attain by age 70.

Modelling the consequences of rejuvenation therapies

Having identified parameters that accurately reproduce real-world data, we then explored how damage and mortality would be postponed by future therapies. For this study we focussed on therapies that remove pre-existing damage (or, equivalently, make it no longer truly damage by eliminating the mechanism whereby it translates into eventual functional decline). This is easy to model in our system: for a single mechanism, we simply set both the existing damage attributable to it and the subsequent rate of occurrence of that damage to zero. The distribution of contributions of each mechanism to its category of damage can thus be defined so as to specify what proportion of a category is addressed with each successive therapy.

The design of our model allows us to set up a schedule of arrival of such therapies, which can be applied to cohorts of various ages. Clearly a wide range of such schedules can be imagined. For this study, we explored a schedule in which the remaining damage in each category is 50% eliminated with each therapy applied to it, and in which such therapies arrive at uniform intervals.

In evaluating the relevance of this simulation to future progress against aging, a key consideration is the predictability of the timeframe of future progress. In our view, the history of technology sends a clear message on this point: that there is a profound difference between fundamental breakthroughs and incremental refinements of those breakthroughs. Specifically, the timeframes for fundamental breakthroughs are very hard to predict, but once those breakthroughs have been made, their cumulative refinement by individually modest advances proceeds at a rather predictable rate so long as public enthusiasm for progress remains strong. Thus, in this study

Fig. 2 a, b Fitting of the simulation to United States mortality data for 1999. *Orange* Actual data. *Yellow* Gompertz fit. *Green* Best-fit simulation. **a** Log(mortality); **b** distribution of age at death. Note: colours appear in online version of this article and in reprints available on request

we factor out the timeframe for initial, fundamental advances in combatting aging and focus on what may occur thereafter. We do this by defining time zero as the date of the first breakthrough—the first point at which one component mechanism of aging is “cured,” i.e. when one category of aging damage is 50% cured.

Figure 4 shows the results of this simulation, plotted as the category-maximum level of damage (DMAX) present in the median individual within a simulation of 100,000 individuals, as a function of age. Each curve represents a cohort of 100,000 individuals of the same age; the first therapy (eliminating half the damage in one category) arrives at age 10 (youngest cohort) through 80 (oldest cohort). In the simulations shown, therapies arrive every 6 years and there are seven categories of damage: thus, within a given category the damage and rate of accumulation of new damage are halved every 42 years. Category-maximum damage accumulation begins to be

affected even after the first intervention (whose timing is identified by the circles), because of the influence of each category’s level on the rates of accumulation of other categories.

As one would expect, it takes more than 36 years from the arrival of the first therapy before the DMAX of any cohort falls back to its pre-therapy level, since until then at least one category of damage has yet to be addressed at all. Notably, however, the older a cohort is when therapies start to arrive, the sooner they return to their pre-treatment level of damage.

Figure 5 shows the consequences of these trajectories of damage level for the probability of survival to a given age assuming no age-independent death. The survival curve for individuals who first receive therapies when already 80 years old differs only slightly from a contemporary curve (black line in Fig. 5), with only a small minority of the cohort obtaining a few additional years of life. This is simply because, even though typical

Fig. 3 Category-maximum damage levels (DMAX) for a representative subset of individuals in the simulation whose mortality behaviour is shown in Fig. 2

members of this cohort never incur a burden of damage sufficient to be certain to kill them outright, they carry a high proportion of that burden for a few decades, during which their annual mortality risk is substantial. This means that the chance of their surviving long enough to benefit from the later therapies is negligible. As few as 10 years of “head start,” however—in other words, access to the first therapies at age 70—gives a 10% chance of surviving long enough that the chance of dying from aging at any age becomes infinitesimal, and the cohort 10 years younger than that has almost a 50% chance of achieving this state of indefinite youth.

Discussion

In this study, we have demonstrated that a simulation built on a biologically realistic model of aging (namely,

as the accumulation of molecular and cellular damage) can very precisely describe the pattern of mortality seen in populations with low age-independent mortality. We have made use of the biologically motivated structure of our model to investigate the potential effects of biomedical progress in combatting this damage, and in particular the effects of a succession of interventions each of which eliminates a component of that damage. This has not been possible using previous models of the aging process and its effects on health and mortality, because informative simulations of the life-extending effects of interventions cannot easily be performed starting from models of aging that lack a sound biological basis.

Our simulation has many parameters, and that fact would in some circumstances be a basis for criticism—the more parameters one includes, the easier it is to find values that give a good fit to one’s data. But in this case it should be noted that all parameters exist for a sound biological reason. (In Einstein’s words: “Everything should be made as simple as possible, but not simpler.”) It is also worth noting that the goodness-of-fit of our simulation to the source data turns out to be extremely insensitive to the values of many of the parameters (data not shown).

The class of intervention we have investigated here is those that “cure” some precursor of aging. Some of the immediate molecular and cellular side-effects of metabolism build up throughout life, even from before birth, and increasingly restrict metabolism’s ability to maintain organ and organismal function. Our simulation can directly model the effect of a therapy that removes such

Fig. 4 Median DMAX levels for cohorts who are aged 10, 20, ... 80 when the first therapy arrives, under the schedule of therapies described in the text. *Circles* Point at which the various cohorts receive their first therapies

Fig. 5 Survival curves for cohorts aged 10, 20, ... 80 when the first therapy arrives, under the schedule of therapies described in the text. *Black line* Survival curve in the absence of any such therapies at any age

a precursor: one of the types of damage is simply set to zero thereafter. It should be stressed that this is a reasonable representation of such therapies across the spectrum of real-life precursors of aging, because there is an equivalence between wholly eliminating one minor contributor to aging and partly eliminating a major one. Thus, for example, it may be possible to halve the age-related increase in stiffness of the artery wall with a drug that cleaves essentially all of one type of molecular cross-link but does not cleave other types, whereas it may be more likely that we would halve the effect of mitochondrial mutations by gene therapy that potentially rescues all cells but in practice reaches only half of them because of the limitations of the vector. These two cases are equivalent for the purposes of our simulation: they are both represented by the elimination of a mechanism that contributes half the damage in a particular category.

Our main finding in this study is that the concept of a “longevity escape velocity,” first introduced by one of us in 2004 (de Grey 2004), is indeed a quantitatively realistic scenario once therapies that eliminate important contributors to aging begin to emerge. Specifically, we show that, if the impact of each category of aging mechanism is halved every ~40 years or faster, beneficiaries of the first round of such therapies will experience a progressive decline in their damage level over the long term, and thus a reduction to negligible levels of their risk of death from age-related causes. Indefinite lifespans can therefore be achieved, even without the (probably unrealistic) total elimination of all age-related damage.

We have also established that longevity escape velocity cannot be achieved by a schedule of cures that

slows down too fast (in terms of the proportion of each category of damage eliminated per unit time). This might seem counterintuitive—surely if aging is being slowed, by the depletion of the contributors to it, we are buying more time to fix the next set of problems than we had for the last set? The explanation is to be found in the parameters that we found fit real-life mortality data: the rate of accumulation of damage is dominated by the feedback component, i.e. by how much damage is already present, rather than the basal (constant) component. Thus, damage for which there is as yet no therapy has approximately a constant doubling time. However, our experiments indicate (data not shown) that this still allows cures to slow down somewhat, to roughly a harmonic rate of progress (in which the remaining amount cured every 40 years is 1/2, 1/3, 1/4... rather than always 1/2): only an exponential deceleration in progress (e.g. 1/2, 1/4, 1/8...) fails.

This study lays the foundation for a wide range of future work. It will be instructive, for example, to determine how rapidly cures would have to be developed if they prevented the further accumulation of a given precursor of aging but did not remove what had already accumulated; we believe that such cures could not achieve longevity escape velocity without a probably unachievable rate of progress, as indicated by the green curve in Fig. 1. Perhaps more importantly, this simulation describes for the first time a concrete goal for the biomedical gerontology profession that can be objectively described as “curing aging” while also being structured in terms that show how working scientists can most effectively contribute to it. It is our hope

that this will help to energise the biomedical community and hasten the development of such therapies.

Conclusion

The main finding of this study is that future advances in treating the underlying substrate of aging—the damage that arises and accumulates throughout life as side-effects of metabolism and eventually overwhelms the ability of metabolism to maintain function—potentially possess a property that one of us has previously termed “longevity escape velocity” (de Grey 2004). Our model predicts that, if progress in eliminating successive types of damage continues

at a sufficient pace (which, we stress, is a pace very typical of past technologies—prominent examples include powered flight, computers and the combatting of infectious diseases), the amount of damage present in the body can be kept low, with a negligible rate of age-related death per year, irrespective of a person’s age. The potential consequences of such therapies for human healthy life expectancy are therefore dramatic and worthy of urgent attention.

Appendix

In addition to the source code in this file, this program uses WINBGIM Version 3.4, 21 December 1999,

LEScon.h

```
#include <cstdlib>
#include <iostream>
#include <winbgim.h>
#include <math.h>
#include <time.h>
#include <iostream>
#include <fstream>
#include "newran03/newran.h"
#define NUM_CATEGORIES 7
#define MAX_MECH_PER_CATEGORY 30
#define MAX_AGE 402
#define MAX_POP 100000
#define MAX_C_AGE 140
#define MAX_C_POP 10000

extern int MECH_PER_CATEGORY;
extern int NORMAL_ABS;
extern int FATAL_LEVEL_LOGNORM;
extern int CONST_FATAL;
extern double MECH_DIVISOR[];
extern double CAT_MULT[];
extern double BASAL_HOMOGENEITY, BASAL_RATE_SD, BASAL_RATE, EXP_RATE, EXP_RATE_SD,
    EXP_HOMOGENEITY, FATAL_LEVEL, FATAL_LEVEL_SD;
extern int WAIT_CHALLENGE;

struct Damage {
    int age;
    double maxDamage;
    double fatalLevel;
};

// A is actuarial (US '99), G is Gompertz
struct STATS {
```

```

        double mean;
        double sd;
        double nsd;
        double skew;
        double kurt;
        double ar2;
        double gr2;
};
extern STATS amStats, gmStats;
#define GMODEL_SPAN 130 /* Max age to run the model */
#define AMODEL_SPAN 113
extern double cDamage [MAX_C_POP][MAX_C_AGE];
void drawGrid(int x, int y);
void plotGomp(double array[], int len, int color, int x, int y);

#define ZERO(var) memset(var, 0, sizeof(var))
extern double gmodel[], amodel[];
extern int POPULATION;
int test(double normYearDeaths[]=NULL, Damage damages[]=NULL, int rawYearDeaths[]=NULL,
         int nCures[]=NULL, int report=0, char *title=NULL);
extern void playwithParams(void);
void playwithChallenge(void);
extern int statStart;
void statistics(double dist[], int len, STATS &s);
void fillModels(void);
double dist(double m, double frac, int lognorm);

```

LEScon.cpp

```

#include "LEScon.h"

int POPULATION=100;
int CURE=0;
int CURE_THRESHOLD=20;
int CURE_SPACE = 5;
int CURE_DELAY = 40;
int CURE_SPACE2 = 5;
int CURE_DELAY2 = 40;
int USECP1 = 7;

```

```

int BASAL_RATE_LOGNORM = 1;
int EXP_RATE_LOGNORM = 1;

using namespace std;
Normal normal;
VariLogNormal vln;
double dist(double m, double frac, int lognorm)
{
    double res;
    if (m == 0)
        return 0;
    if (lognorm) {
        res = vln.Next(m, frac*m);
    }
    else {
        res = (normal.Next() * frac*m + m);
        if (NORMAL_ABS) res = fabs(res);
    }
    return res;
}

double gl_myBasal, gl_myExp;
class Mechanism {
public:
    int id;
    int cured;
    double slowFactor;
    double damage;
    double basalRate;
    double expRate;
    Mechanism(int myId, double mySlowFactor);
    double tick(double avDamage);
    void cure(void);
};
Mechanism::Mechanism(int myId, double mySlowFactor)
{
    id = myId;
    slowFactor = mySlowFactor;
    cured = 0;
    damage = 0; //dist(INIT_DAMAGE, INIT_DAMAGE_SD, INIT_DAMAGE_LOGNORM)
    basalRate = dist(BASAL_RATE, BASAL_RATE_SD, BASAL_RATE_LOGNORM);
    basalRate = basalRate*(1-BASAL_HOMOGENEITY) + gl_myBasal*BASAL_HOMOGENEITY;
}

```

```

    expRate = dist(EXP_RATE, EXP_RATE_SD, EXP_RATE_LOGNORM);
    expRate = expRate*(1-EXP_HOMOGENEITY) + gl_myExp*EXP_HOMOGENEITY;
}
double Mechanism::tick(double avDamage)
{
    if (!cured) {
        double newDamage = basalRate;
        newDamage += expRate*avDamage;
        damage += newDamage / slowFactor;
    }
    return damage;
}
void Mechanism::cure(void)
{
    cured = 1;
    damage = 0;
}
class Category
{
public:
    int id;
    double damage;
    Mechanism *mechanisms[MAX_MECH_PER_CATEGORY];
    Category(int myId);
    ~Category();
    double tick(double avDamage);
};
#define ITER_MECH int mechn; Mechanism **mechp; for (mechn=0,mechp=&mechanisms[0];
mechn<MECH_PER_CATEGORY; mechn++, mechp=&mechanisms[mechn])
Category::Category(int myId)
{
    id = myId;
    damage = 0;
    ITER_MECH {
        *mechp = new Mechanism(id*1
                                000+mechn, MECH_DIVISOR[mechn]);
    }
}
Category::~~Category()
{
    ITER_MECH {
        delete *mechp;
    }
}

```

```

}
double Category::tick(double avDamage)
{
    damage = 0;
    ITER_MECH {
        damage += (*mechp)->tick(avDamage);
    }
    return damage;
}

class Person
{
public:
    Category *categories[NUM_CATEGORIES];
    double damage, maxDamage, fatalLevel;
    int age;
    int dead;

    Person(void);
    ~Person();
    int autopsy(void);
    int tick(void);
    int cure(int id);
};

#define ITER_CAT int catn; Category **catp; for (catn=0, catp=&categories[0];
catn<NUM_CATEGORIES; catn++, catp=&categories[catn])
Person::Person(void)
{
    gl_myBasal = dist(BASAL_RATE, BASAL_RATE_SD, BASAL_RATE_LOGNORM);
    gl_myExp = dist(EXP_RATE, EXP_RATE_SD, EXP_RATE_LOGNORM);
    ITER_CAT {
        *catp = new Category(catn);
    }
    damage = 0; maxDamage = 0; fatalLevel = 0;
    age = 0;
    dead = 0;
}

Person::~~Person()
{
    ITER_CAT {
        delete *catp;
    }
}

Uniform U; // uniform random, range 0..1
int Person::tick(void)

```



```

{
    double catDamage, sumDamage = 0;
    if (dead)
        return 1;
    age++;
    maxDamage = 0;
    ITER_CAT {
        catDamage = (*catp)->tick(damage) * CAT_MULT[catn];
        sumDamage += catDamage;
        maxDamage = fmax(maxDamage, catDamage);
    }
    if (CONST_FATAL) {
        fatalLevel = U.Next()*FATAL_LEVEL;
    } else {
        fatalLevel = dist(FATAL_LEVEL, FATAL_LEVEL_SD, FATAL_LEVEL_LOGNORM);
    }
    if (maxDamage > fatalLevel && !WAIT_CHALLENGE) {
        dead = 1;
    }
    damage = sumDamage/NUM_CATEGORIES;
    if (dead)
        return 1;
    return 0;
}
int Person::autopsy(void)
{
    /*This only works if the categories are all compared to the same
    # number (as each other) each year to see whether damage is fatal.
    # It doesn't remember which category killed the
    # person, just looks anew for which category has the highest
    # damage.*/
    double dam, hiDam = 0;
    Category *hiCat = NULL;
    ITER_CAT {
        dam = (*catp)->damage;
        if (dam > hiDam) {
            hiDam = dam;
            hiCat = *catp;
        }
    }
    hiDam = 0;
    if (!hiCat) return -1;
    Mechanism **mechanisms = hiCat->mechanisms;
    Mechanism *hiMech = NULL;

```

```

    ITER_MECH {
        dam = (*mechp)->damage;
        if (dam > hiDam) {
            hiDam = dam;
            hiMech = *mechp;
        }
    }
    if (!hiMech) return -1;
    return hiMech->id;
}
int Person::cure(int id)
{
    int catn = id/1000; int mechn = id%1000;
    categories[catn]->mechanisms[mechn]->cure();
}

class Population
{
public:
    Person *people[MAX_POP];
    Person *deaths[MAX_POP];
    int nDeaths;

    Population(void);
    ~Population();
    int tick(Person ***deathsArray, int age);
    void cure(int id);
};

#define ITER_PERSON int pn; Person **pp; for (pn = 0, pp = &people[0]; pn < POPULATION;
pn++,pp=&people[pn])
Population::Population(void)
{
    nDeaths = 0;
    ITER_PERSON {
        *pp = new Person;
    }
}

Population::~~Population()
{
    ITER_PERSON {
        delete *pp;
    }
}

int Population::tick(Person ***deathsArray, int age)
{

```

```

    *deathsArray = &deaths[nDeaths];
    ITER_PERSON {
        if (!(*pp)->dead) {
            int dead = (*pp)->tick();
            if (dead) {
                deaths[nDeaths] = *pp;
                nDeaths++;
            }
        }
        if (WAIT_CHALLENGE) {
            cDamage[pn][age] = (*pp)->damage;
        }
    }
    return &deaths[nDeaths]-*deathsArray;
}

void Population::cure(int id)
{
    ITER_PERSON
        if (!(*pp)->dead)
            (*pp)->cure(id);
}

int test(double normYearDeaths[], Damage damages[], int rawYearDeaths[], int nCuresA[],
        int report, char *title)
{
    Population p;
    int age;
    int nCures=0;
    Person *pDeaths;
    int nYearDeaths[MAX_AGE+1];
    Person **pYearDeaths[MAX_AGE+1];
    int peopleKilled[NUM_CATEGORIES][MAX_MECH_PER_CATEGORY];    // Used in CURE==2
    memset(peopleKilled, 0, sizeof(peopleKilled));
    for (age = 0; age < (WAIT_CHALLENGE ? MAX_C_AGE : MAX_AGE); age++) {
        //printf("Age %d, damage %f\n", age, p.people[0]->damage);
        int nd;
        nd = nYearDeaths[age] = p.tick(&(pYearDeaths[age]), age);
        if (nd) {
            ;//printf("Age %d, deaths %d\n", age, nd);
        }
        if (CURE == 1) {
            if (age >= CURE_DELAY && ((age-CURE_DELAY) % CURE_SPACE) == 0) {
                int cnum = (age-CURE_DELAY)/CURE_SPACE;

```

```

        if (cnum < NUM_CATEGORIES*(MECH_PER_CATEGORY-1)) {
            int cure=(cnum%NUM_CATEGORIES)*1000+(cnum/NUM_CATEGORIES);
            printf("AGE %d cure %d ",age, cure);
            p.cure(cure);
            nCures++;
            if (nCures%7 == 0)
                ;//printf("\n");
        }
    }
}
if (CURE == 2) {
#define KNUM(cause) peoplekilled[cause/1000][cause%1000]
    for (int i = 0; i < nd; i++) {
        int cause = pYearDeaths[age][i]->autopsy();
        if (cause == -1) continue;
        KNUM(cause)++;
        if (KNUM(cause) == CURE_THRESHOLD) {
            p.cure(cause);
            printf("age %d cure %d ",age, cause);
            nCures++;
            if (nCures%7 == 0)
                printf("\n");
        }
    }
    if (nCuresA) {
        nCuresA[age] = nCures;
    }
}
if (CURE == 3) {
    if (age >= CURE_DELAY && ((age-CURE_DELAY) % CURE_SPACE) == 0) {
        int mechnum = (age-CURE_DELAY)/CURE_SPACE;
        if (mechnum < (MECH_PER_CATEGORY-1)) {
            for (int catnum=0; catnum<USECP1; catnum++) {
                int cure=catnum*1000+mechnum;
                printf("AGE %d cure1 %d catnum %d cp1 %d\n",age, cure, catnum,
USECP1);

                p.cure(cure);
                nCures++;
                if (nCures%7 == 0)
                    printf("\n");
            }
        }
    }
    if (age >= CURE_DELAY2 && ((age-CURE_DELAY2) % CURE_SPACE2) == 0) {
        int mechnum = (age-CURE_DELAY2)/CURE_SPACE2;
        if (mechnum < (MECH_PER_CATEGORY-1)) {

```

```

        for (int catnum=USECP1; catnum<NUM_CATEGORIES; catnum++) {
            int cure=catnum*1000+mechnum;
            printf("AGE %d cure2 %d ",age, cure);
            p.cure(cure);
            nCures++;
            if (nCures%7 == 0)
                printf("\n");
        }
    }
}
if (p.nDeaths == POPULATION)
    break;
}
//printf("Done with age loop\n");
if (normYearDeaths)
    for (int i=0; i<=age; i++) {
        normYearDeaths[i] = (double)nYearDeaths[i]/POPULATION;
    }
if (rawYearDeaths)
    for (int i=0; i<=age; i++) {
        rawYearDeaths[i] = nYearDeaths[i];
    }
if (damages)
    for (int i=0; i < POPULATION; i++) {
        damages[i].age = p.people[i]->age;
        damages[i].maxDamage = p.people[i]->maxDamage;
        damages[i].fatalLevel = p.people[i]->fatalLevel;
    }
return age;
}

void ConstCureSpacing(void)
{
#define NDEL 20
#define NSP 8
#define SPINC 1
#define DELINC 5
#define AGEINC 50
#define CSP(n) ((n+1)*SPINC)
#define CD(n) ((n+1)*DELINC)
    int rawYearDeaths[NSP][NDEL][MAX_AGE+1];    ZERO(rawYearDeaths);

```



```

int nCures[NSP][NDEL][MAX_AGE+1]; ZERO(nCures);
int oldest[NSP][NDEL], mostOldest=0;
POPULATION=10000;
CURE = 1;
for (int sp=0; sp<NSP; sp++) {
    for (int del=0; del<NDEL; del++) {
        CURE_SPACE = CSP(sp);
        CURE_DELAY = CD(del);
        printf("Testing space %d delay %d\n", CURE_SPACE, CURE_DELAY);
        oldest[sp][del] = test(NULL, NULL, rawYearDeaths[sp][del], nCures[sp][del]);
        mostOldest = max(mostOldest, oldest[sp][del]);
    }
}
ofstream outfile ("c:/ages.csv", ios::trunc);
outfile<<"age,";
for (int sp=0; sp<NSP; sp++) {
    for (int del=0; del<NDEL; del++) {
        outfile<<"s"<<(sp+1)*SPINC<<"d"<<(del+1)*DELINC<<',';
    }
}
outfile<<'\n';
int sum[NSP][NDEL]; ZERO(sum);
for (int age=0; age<mostOldest; age++) {
    outfile<<age<<',';
    for (int sp=0; sp<NSP; sp++) {
        for (int del=0; del<NDEL; del++) {
            sum[sp][del] += rawYearDeaths[sp][del][age];
            outfile << sum[sp][del]<<',';
        }
    }
    outfile << '\n';
}
outfile.close();
outfile.open("c:/deaths.csv", ios::trunc);
int asum[NSP][NDEL][MAX_AGE];
for (int del=0; del<NDEL; del++) {
    for (int sp=0; sp<NSP; sp++) {
        asum[sp][del][0] += rawYearDeaths[sp][del][0];
    }
}
for (int age = 1; age < MAX_AGE; age++) {
    for (int del=0; del<NDEL; del++) {
        for (int sp=0; sp<NSP; sp++) {
            asum[sp][del][age] = rawYearDeaths[sp][del][age] + asum[sp][del][age-1];
        }
    }
}

```

```

    }
  }
}
for (int age = AGEINC; age < MAX_AGE; age+=AGEINC) {
  outfile<<"A:"<<age<<',';
  for (int del=0; del<NDEL; del++) {
    outfile<<"d"<<CD(del)<<',';
  }
  outfile<<'\\n';
  for (int sp=0; sp<NSP; sp++) {
    outfile<<"s"<<CSP(sp)<<',';
    for (int del=0; del<NDEL; del++) {
      outfile<<asum[sp][del][age]<<',';
    }
    outfile<<'\\n';
  }
  outfile<<'\\n';
}
for (int del=0; del<NDEL; del++) {
  outfile<<"D:"<<CD(del)<<',';
  for (int age = AGEINC; age < MAX_AGE; age+=AGEINC) {
    outfile<<"a"<<age<<',';
  }
  outfile<<'\\n';
  for (int sp=0; sp<NSP; sp++) {
    outfile<<"s"<<CSP(sp)<<',';
    for (int age = AGEINC; age < MAX_AGE; age+=AGEINC) {
      outfile<<asum[sp][del][age]<<',';
    }
    outfile<<'\\n';
  }
  outfile<<'\\n';
}
for (int sp=0; sp<NSP; sp++) {
  outfile<<"S:"<<CSP(sp)<<',';
  for (int age = AGEINC; age < MAX_AGE; age+=AGEINC) {
    outfile<<"a"<<age<<',';
  }
  outfile<<'\\n';
  for (int del=0; del<NDEL; del++) {
    outfile<<"d"<<CD(del)<<',';
    for (int age = AGEINC; age < MAX_AGE; age+=AGEINC) {
      outfile<<asum[sp][del][age]<<',';
    }
    outfile<<'\\n';
  }
}

```

```

    }
    outfile<<'\\n';
}
outfile.close();
}

void ConstDeaths(void)
{
#define SDIVS 10
    int rawYearDeaths[SDIVS][MAX_AGE+1];    ZERO(rawYearDeaths);
    int nCures[SDIVS][MAX_AGE+1]; ZERO(nCures);
    int oldest[SDIVS], mostOldest=0, thresh[SDIVS];
    POPULATION=20000;
    CURE = 2;
    for (int i=0; i<SDIVS; i++) {
        CURE_THRESHOLD = POPULATION*i/(35*(SDIVS/2));
        thresh[i] = CURE_THRESHOLD;
        printf("Cure threshold is %d\\n",
               CURE_THRESHOLD, 35*CURE_THRESHOLD, POPULATION);
        oldest[i] = test(NULL, NULL, rawYearDeaths[i], nCures[i]);
        mostOldest = max(mostOldest, oldest[i]);
    }
    ofstream outfile ("c:/ages.csv", ios::trunc);
    for (int col=0; col<SDIVS; col++) {
        outfile<<"Th:"<<thresh[col]<<',';
    }
    for (int col=0; col<SDIVS; col++) {
        outfile <<"Cu:"<<col<<',';
    }
    outfile<<'\\n';
    int sum[SDIVS]; ZERO(sum);
    for (int row=0; row<mostOldest; row++) {
        for (int col=0; col<SDIVS; col++) {
            sum[col] += rawYearDeaths[col][row];
            outfile << sum[col]<<',';
        }
        for (int col=0; col<SDIVS; col++) {
            outfile << nCures[col][row]*POPULATION/35<<',';
        }
        outfile << '\\n';
    }
    outfile.close();
}
}

```

```

void BlendSchedules(void)
{
    CURE_SPACE = 10;
    CURE_DELAY = 40;
    CURE_SPACE2 = 15;
    CURE_DELAY2 = 60;
#define NTRIALS 8
    int rawYearDeaths[NTRIALS][MAX_AGE+1];    ZERO(rawYearDeaths);
    int oldest[NTRIALS], mostOldest=0;
    POPULATION=4000;
    CURE = 3;
    for (int trial=0; trial < NTRIALS; trial++) {
        USECP1 = NUM_CATEGORIES-trial;
        oldest[trial] = test(NULL, NULL, rawYearDeaths[trial]);
        printf("Done with trial, cp1 was %d\n",USECP1);
        mostOldest = max(mostOldest, oldest[trial]);
    }
    ofstream outfile ("c:/ages.csv", ios::trunc);
    for (int trial=0; trial < NTRIALS; trial++) {
        outfile<<"t"<<trial<<',';
    }
    outfile<<'\n';
    int sum[NTRIALS]; ZERO(sum);
    for (int row=0; row<mostOldest; row++) {
        for (int trial=0; trial < NTRIALS; trial++) {
            sum[trial] += rawYearDeaths[trial][row];
            outfile << sum[trial]<<',';
        }
        outfile << '\n';
    }
    outfile.close();
}

int main(int argc, char *argv[])
{
    int t=time(NULL);
    double d=(double)t/2000000000.0;
    printf("TimeSeed %d %f\n",t,d);
    //MT mt(d);
    MotherOfAll mt(d);
    Random::Set(mt);
    try {
        playwithParams();
    }
}

```

```

        playwithChallenge();
        //ConstCureSpacing();
        //ConstDeaths();
        //BlendSchedules();
    }
    catch (...) {
        cout <<"Oops! Exception!!!\n";
        system("PAUSE");
        throw;
    }
    cout<<"\n";
    system("PAUSE");
    return EXIT_SUCCESS;/**/
}

```

vary.cpp

```

#include "LEScon.h"
#define ARLEN(a) (sizeof(a)/sizeof(a[0]))
double *varyVars[] = {&BASAL_HOMOGENEITY, &BASAL_RATE_SD, &BASAL_RATE, &EXP_RATE,
&EXP_RATE_SD,
    &EXP_HOMOGENEITY, &FATAL_LEVEL, &FATAL_LEVEL_SD};
char *varyNames[] = {"BASAL_HOMOGENEITY", "BASAL_RATE_SD", "BASAL_RATE", "EXP_RATE",
"EXP_RATE_SD",
    "EXP_HOMOGENEITY", "FATAL_LEVEL", "FATAL_LEVEL_SD"};
Damage damages[MAX_POP];
void vary(void)
{
    double m,sd,sk,kr;
    //double varyMults[] = {0.9, 0.98, 1.02, 1.1};
    double varyMults[] = {0.5, 0.75, 0.9, 1.0, 1.1, 1.25, 1.5, 2.0};
    //double *varyNames[] = {&FATAL_LEVEL_SD};
    //double varyMults =
{0.375};//,0.125,0.25,0.375,0.5,0.625,0.75,0.875,1.0,1.125,1.25,1.375};
    for (int v=0; v<ARLEN(varyVars); v++) {
        double *pD = varyVars[v];
        double varsave = *pD;
        char label[100];
        for (int vm=0; vm<ARLEN(varyMults); vm++) {
            *pD = varsave * varyMults[vm];

```



```

        sprintf(label, "%s=%.4g", varyNames[v], *pD);
        double normYearDeaths[MAX_AGE+1];
        int oldest=test(normYearDeaths, damages);
        STATS s;
        statistics(normYearDeaths, oldest, s);
        printf("%23s    %3.4f    %3.4f    %3.4f    %3.4f    %3.4f\n", label,
                s.mean, s.sd, s.nsd, s.skew, s.kurt);
    }
    *pD = varsave;
    cout << '\n';
}

}

#define GH 200
#define CX 300
#define CY 300
#define DX 10
#define DY 510
#define LX 160
#define LY 510
#define GOMPX 310
#define GOMPY 510
void drawGrid(int x, int y)
{
    setcolor(COLOR(128,128,0));
    line(x,y,x+140,y);
    line(x,y-GH,x+140,y-GH);
    for (int i = 0; i <= 7; i++) {
        line(x+i*20,y,x+i*20,y-GH);
    }
}

void plotGomp(double array[], int len, int color, int x, int y)
{
    double sum=array[0];
    for (int i=1; i<len; i++) {
        double gval = log(array[i]/(1-sum));
        putpixel(i+x, (int)(y-150-gval*15), color);
        sum += array[i];
    }
}

void plotResults(int oldest, double normYearDeaths[], Damage damages[])
{
    drawGrid(CX,CY);
    drawGrid(DX,DY);
    outtextxy(DX, DY+30, "Max Damage");
}

```

```

drawGrid(LX,LY);
outtextxy(LX, LY+30, "Fatal Level");
setcolor(COLOR(128,255,0));
for (int i = 1; i < AMODEL_SPAN; i++) {
    line(i+CX, (int)(CY-amodel[i]*GH*30), i-1+CX, (int)(CY-amodel[i-1]*GH*30));
}
setcolor(COLOR(255,128,0));
for (int i = 1; i < GMODEL_SPAN; i++) {
    line(i+CX, (int)(CY-gmodel[i]*GH*30), i-1+CX, (int)(CY-gmodel[i-1]*GH*30));
}
for (int i = 0; i < oldest; i++) {
    putpixel (i+CX, (int)(CY-normYearDeaths[i]*GH*30), COLOR(255,0,255));
}
for (int i = 0; i < POPULATION; i++) {
    putpixel(DX+damages[i].age, (int)(DY-damages[i].maxDamage*GH),
COLOR(255,255,255));
    putpixel(LX+damages[i].age, (int)(LY-damages[i].fatalLevel*GH),
COLOR(255,255,255));
}
setcolor(COLOR(128,128,0));
line(GOMPX, GOMPY, GOMPX, GOMPY-150);
line(GOMPX, GOMPY, GOMPX+120, GOMPY);
plotGomp(gmodel, GMODEL_SPAN, COLOR(128,255,0), GOMPX, GOMPY);
plotGomp(amodel, AMODEL_SPAN, COLOR(255,128,0), GOMPX, GOMPY);
plotGomp(normYearDeaths, oldest, COLOR(200,200,255), GOMPX, GOMPY);
}

#define VPTS 5/*should be odd*/
STATS stats[VPTS*20];
int vLen=ARLEN(varyVars);
double tweaks[20];

void doVary(int pos)
{
    int savePop = POPULATION;
    POPULATION /= 5;
    double saveVar = *varyVars[pos];
    double tval = 1-tweaks[pos];
    *varyVars[pos] = saveVar*pow(tval, VPTS/2);
    double normYearDeaths[MAX_AGE+1];
    double m,sd,sk,kr;
    for (int i = 0; i < VPTS; i++) {
        int oldest = test(normYearDeaths, NULL);
        statistics(normYearDeaths, oldest, stats[i+pos*VPTS]);
        if (i == 1) printf("pos %d stpos %d mean %f\n",pos, i+pos*VPTS, m);
    }
}

```

```

        *varyVars[pos] /= tval;
    }
    POPULATION = savePop;
    *varyVars[pos] = saveVar;
/*      for (int j=0; j<vLen; j++) {
        printf("Done  %s = %.8f  ",varyNames[j],*varyVars[j]);
        } printf("\n\n");*/
}

#define VX (480+150*(i/4))
#define VY 120
double pScale = 1;
int curVar = 0;
void plotVary(bool actuarial)
{
#define S(o,var) stats[i*VPTS+o].var
#define CLIP(exp) min(max((int)(exp),y-VY/3),y+VY/3)
#define MEANM 1
#define NSDM 80
#define SKEWM 16
#define KURTM 4
#define R2M 80
    for (int i = 0; i < vLen; i++) {
        int y=VY*i+VY/2+100-VY*4*(i/4);
        char label[]="  a";
        setcolor(COLOR(255,255,255));
        line(VX, y, VX+100, y);
        label[0] = (i==curVar)?'*':' ';
        label[2] = 'a'+i;
        outtextxy(VX-25,y-8,label);
        setcolor(COLOR(64,64,128));
        line(VX+20,y+VY/2,VX+80,y+VY/2);
        for (int j=0; j<VPTS; j++) {
            line(VX+j*100/(VPTS-1), y+VY/3, VX+j*100/(VPTS-1), y-VY/3);
        }
#define LINES(r,g,b,mvar,svar,mul) \
        setcolor(COLOR(r,g,b)); \
        for (int j = 0; j < VPTS-1; j++) { \
            line(VX+100/(VPTS-1)*j, CLIP(y+pScale*mul*(mvar-S(j,svar))), \
                VX+100/(VPTS-1)*(j+1), CLIP(y+pScale*mul*(mvar-S(j+1,svar))));}
        STATS mStats;
        if (actuarial) {
            mStats = amStats;
        } else {

```

```

        mStats = gmStats;
    }
    printf("i %d  nsd_val %f mnsd %f clip %d  ", i, S(1, nsd), mStats.nsd,
CLIP(NSDM*(y+mStats.nsd-S(1, nsd)))-y);
    LINES(255, 0, 0, mStats.mean, mean, MEANM)
    LINES(255, 255, 0, mStats.nsd, nsd, NSDM)
    LINES(0, 255, 0, mStats.skew, skew, SKEWM)
    LINES(0, 100, 255, mStats.kurt, kurt, KURTM)
    if (actuarial) {
        LINES(128, 0, 128, 1, ar2, R2M)
    } else {
        LINES(128, 0, 128, 1, gr2, R2M)
    }
}
}
char screen[800*600*4+4];
void writeScreen(void)
{
    ofstream outfile ("c:/ascreen.raw", ios::trunc|ios::binary);
    getimage(0, 0, 799, 599, screen);
    for (int y=0; y<600; y++) {
        for (int x=0; x<800; x++) {
            int c=getpixel(x,y);
            outfile.write(&screen[y*3200+x*4], 3);
            //outfile <<(unsigned char)(RED_VALUE(c))<<(unsigned
char)GREEN_VALUE(c)<<(unsigned char)BLUE_VALUE(c);
        }
    }
    outfile.close();
}
void playwithParams(void)
{
    bool actuarial = 1;
    fillModels();
    printf("%23s    Mean,        SD,        normSD,        Skew,    kurtosis\n", " ");
    printf("Pop=%d\n", POPULATION);
    initwindow(800, 600);
    settextstyle(9, HORIZ_DIR, 0);
    setusercharsize(1, 3, 1, 3);
    for (int i=0; i<vLen; i++)
        tweaks[i] = 0.2;
    int reTest = 1;
    int quit=0;
    double normYearDeaths[MAX_AGE+1];

```

```

int oldest;
for (int i=0; i<vLen; i++)
    doVary(i);
STATS s;
while(!quit) {
    if (reTest) {
        oldest = test(normYearDeaths, damages);
    }
    doVary(curVar);
    cleardevice();
    plotVary(actuarial);
    int prPos = 10;
    char outstr[200];
    setcolor(COLOR(255,255,0));
    for (int i=0; i<vLen; i++) {
        sprintf(outstr, "%c %c: %s = %f", i==curVar?'*':' ', 'a'+i,
            varyNames[i], *varyVars[i]);
        outtextxy(0, prPos, outstr);
        prPos += 20;
    }
    sprintf(outstr, "Population = %d", POPULATION);
    outtextxy(0, prPos, outstr); prPos += 20;
    sprintf(outstr, "Tweak = %f", tweaks[curVar]);
    outtextxy(0, prPos, outstr); prPos += 20;
    outtextxy(0, prPos, actuarial?"Actuarial":"Gompertz"); prPos += 20;
    sprintf(outstr, "Stat Range Start = %d", statStart);
    outtextxy(0, prPos, outstr); prPos += 20;
    sprintf(outstr, "%23s    Mean,    normSD,    Skew, Kurtosis, ar^2,    gr^2", " ");
#define TX CX+40
    outtextxy(TX, 20, outstr);
    sprintf(outstr, "%23s    %3.4f    %3.4f    %3.4f    %3.4f    %3.4f", "Actuar",
        amStats.mean, amStats.nsd, amStats.skew, amStats.kurt,
        amStats.ar2, amStats.gr2);
    outtextxy(TX, 40, outstr);
    sprintf(outstr, "%23s    %3.4f    %3.4f    %3.4f    %3.4f    %3.4f", "Gomper",
        gmStats.mean, gmStats.nsd, gmStats.skew, gmStats.kurt,
        gmStats.ar2, gmStats.gr2);
    outtextxy(TX, 60, outstr);
    statistics(normYearDeaths, oldest, s);
    sprintf(outstr, "%23s    %3.4f    %3.4f    %3.4f    %3.4f    %3.4f", "Sim",
        s.mean, s.nsd, s.skew, s.kurt, s.ar2, s.gr2);
    outtextxy(TX, 80, outstr);
    plotResults(oldest, normYearDeaths, damages);
    char ch = getch();

```



```

setcolor(COLOR(255,255,255));
outtextxy(TX+80,100,"WAIT....");
reTest = 0;
if (ch >= 'a' && ch < 'a'+vLen) {
    curVar = ch-'a';
    continue;
} else switch(ch) {
    case 'P': if (POPULATION > 1) POPULATION /= 2; reTest = 1; continue;
    case 'p': POPULATION *= 2; reTest = 1; continue;
    case '2': tweaks[curVar] *= 2; continue;
    case '1': tweaks[curVar] /= 2; continue;
    case ' ': reTest = 1; continue;
    case '4': *varyVars[curVar] /= 1-tweaks[curVar]; reTest=1; continue;
    case '3': *varyVars[curVar] *= 1-tweaks[curVar]; reTest=1; continue;
    case '.': for (int i=0; i<vLen; i++) doVary(i); continue;
    case 's': pScale *= 2; continue;
    case 'S': pScale /= 2; continue;
    case 'w': writeScreen(); continue;
    case 'q': quit=1; continue;
    case 'A': actuarial = 1; continue;
    case 'G': actuarial = 0; continue;
    case 'r': if (statStart >= 20) statStart -= 20;
               statistics(amodel, AMODEL_SPAN, amStats);
               statistics(gmodel, GMODEL_SPAN, gmStats);
               continue;
    case 'R': if (statStart <= 80) statStart += 20;
               statistics(amodel, AMODEL_SPAN, amStats);
               statistics(gmodel, GMODEL_SPAN, gmStats);
               continue;
}
}
printf("\n\n");
printf("//      Mean,      normSD,      Skew, Kurtosis, ar^2, gr^2\n");
printf("//Actuar %3.4f %3.4f %3.4f %3.4f %3.4f %3.4f\n",
        amStats.mean, amStats.nsd, amStats.skew, amStats.kurt,
        amStats.ar2, amStats.gr2);
printf("//Gomper %3.4f %3.4f %3.4f %3.4f %3.4f %3.4f\n",
        gmStats.mean, gmStats.nsd, gmStats.skew, gmStats.kurt,
        gmStats.ar2, gmStats.gr2);
printf("//Sim %3.4f %3.4f %3.4f %3.4f %3.4f %3.4f\n",
        s.mean,s.nsd,s.skew,s.kurt, s.ar2, s.gr2);
printf("//Stat Range = %d to end\n",statStart);
printf("// Population = %d\n",POPULATION);
cout<<'\\n';

```

```

    for (int i=0; i<vLen; i++) {
        printf("double %s = %.8f;\n", varyNames[i], *varyVars[i]);
    }
    closegraph();          //close graphics window
}

```

params.cpp

```

#include <assert.h>

#include "LEScon.h"
/* Canonical EXP_RATE = pow(2, (1.0/20))-1 */
/* const - good for Gompertz */
int MECH_PER_CATEGORY = 5;
double MECH_DIVISOR[] = {1,2,4,8,8};
//double MECH_DIVISOR[] = {2.5,2.5,2.5,2.5,2.5};
double CAT_MULT[NUM_CATEGORIES] = {1,1,1,1,1,1,1};
int FATAL_LEVEL_LOGNORM = 0;
int NORMAL_ABS = 0;
int CONST_FATAL = 1;

//      Mean,      normSD,      Skew,      Kurtosis,      ar^2,      gr^2
//Actuar 78.7283    0.1977    -1.8408    8.4616    1.0000    0.9876
//Gomper 80.0171    0.1722    -1.0700    4.8656    0.9876    1.0000
//Sim     79.4792    0.1741    -0.8564    4.2376    0.9690    0.9850
//Stat Range = 0 to end
// Population = 12800

double BASAL_HOMOGENEITY = 0.45000000;
double BASAL_RATE_SD = 0.20000000;
double BASAL_RATE = 0.00000069;
double EXP_RATE = 0.04898692;
double EXP_RATE_SD = 0.17777778;
double EXP_HOMOGENEITY = 0.32400000;
double FATAL_LEVEL = 0.54000000;
double FATAL_LEVEL_SD = 0.33755274;
/**/

/* normal reflected - good fit*/
int MECH_PER_CATEGORY = 5;
double MECH_DIVISOR[] = {1,2,4,8,8};
//double MECH_DIVISOR[] = {2.5,2.5,2.5,2.5,2.5};
double CAT_MULT[NUM_CATEGORIES] = {1,1,1,1,1,1,1};

```

```

int FATAL_LEVEL_LOGNORM = 0;
int NORMAL_ABS = 1;
int CONST_FATAL = 0;

//      Mean,      normSD,      Skew,      Kurtosis,      ar^2,      gr^2
//Actuar 79.5640    0.1695    -1.1345    4.6997    1.0000    0.9907
//Gomper 80.1367    0.1684    -0.9416    4.1777    0.9907    1.0000
//Sim    79.5049    0.1769    -1.0856    4.7310    0.9846    0.9864
//Stat Range = 20 to end
// Population = 32000

double BASAL_HOMOGENEITY = 0.45000000;
double BASAL_RATE_SD = 0.20000000;
double BASAL_RATE = 0.00019073;
double EXP_RATE = 0.02445427;
double EXP_RATE_SD = 0.27777778;
double EXP_HOMOGENEITY = 0.32400000;
double FATAL_LEVEL = 0.90000000;
double FATAL_LEVEL_SD = 0.33755274;
/**/

/* High basal; poor with or without refl norm, impossible with lognorm* /
int MECH_PER_CATEGORY = 5;
double MECH_DIVISOR[] = {1,2,4,8,16};
double CAT_MULT[NUM_CATEGORIES] = {1,1,1,1,1,1,1};
int FATAL_LEVEL_LOGNORM = 0;
int NORMAL_ABS = 1;
int CONST_FATAL = 0;

//      Mean,      normSD,      Skew,      Kurtosis,      ar^2,      gr^2
//Actuar 78.7283    0.1977    -1.8408    8.4616    1.0000    0.9876
//Gomper 80.0171    0.1722    -1.0700    4.8656    0.9876    1.0000
//Sim    79.3743    0.1742    -0.7714    4.0096    0.9636    0.9769
//Stat Range = 0 to end
// Population = 4000

double BASAL_HOMOGENEITY = 0.51200000;
double BASAL_RATE_SD = 0.02147484;
double BASAL_RATE = 0.00317637;
double EXP_RATE = 0.00444100;
double EXP_RATE_SD = 0.22222222;
double EXP_HOMOGENEITY = 0.40500000;
double FATAL_LEVEL = 1.25000000;
double FATAL_LEVEL_SD = 0.25263158;/**/

```

```

/* absnorm, big basal - lousy* /
int MECH_PER_CATEGORY = 5;
double MECH_DIVISOR[] = {1,2,4,8,8};
//double MECH_DIVISOR[] = {2.5,2.5,2.5,2.5,2.5};
double CAT_MULT[NUM_CATEGORIES] = {1,1,1,1,1,1,1};
int FATAL_LEVEL_LOGNORM = 1;
int NORMAL_ABS = 0;
int CONST_FATAL = 0;

//      Mean,      normSD,      Skew,      Kurtosis,      r^2
//Ideal  80.0171   0.1722   -1.0700   4.8656   1.0000
//Sim    82.7918   0.1162   0.1031   2.9161   0.9157
// Population = 1000

double BASAL_HOMOGENEITY = 0.45000000;
double BASAL_RATE_SD = 0.12800000;
double BASAL_RATE = 0.00215579;
double EXP_RATE = 0.00972128;
double EXP_RATE_SD = 0.22222222;
double EXP_HOMOGENEITY = 0.79101563;
double FATAL_LEVEL = 1.38504155;
double FATAL_LEVEL_SD = 0.31536879;
/**/

/* lognorm, big basal - mediocre fit (bad skew)* /
int MECH_PER_CATEGORY = 5;
double MECH_DIVISOR[] = {1,2,4,8,8};
//double MECH_DIVISOR[] = {2.5,2.5,2.5,2.5,2.5};
double CAT_MULT[NUM_CATEGORIES] = {1,1,1,1,1,1,1};
int FATAL_LEVEL_LOGNORM = 1;
int NORMAL_ABS = 0;
int CONST_FATAL = 0;

//      Mean,      normSD,      Skew,      Kurtosis,      r^2
//Ideal  80.0171   0.1722   -1.0700   4.8656   1.0000
//Sim    83.2823   0.1306   -0.0908   2.9140   0.9742
// Population = 8000

double BASAL_HOMOGENEITY = 0.45000000;
double BASAL_RATE_SD = 0.12800000;
double BASAL_RATE = 0.00526316;
double EXP_RATE = 0.00005541;
double EXP_RATE_SD = 0.22222222;
double EXP_HOMOGENEITY = 0.79101563;
double FATAL_LEVEL = 1.38504155;

```

```

double FATAL_LEVEL_SD = 0.25229503;
/**/

/* normal non-reflected - mediocre fit* /
int MECH_PER_CATEGORY = 5;
double MECH_DIVISOR[] = {1,2,4,8,8};
//double MECH_DIVISOR[] = {2.5,2.5,2.5,2.5,2.5};
double CAT_MULT[NUM_CATEGORIES] = {1,1,1,1,1,1,1};
int FATAL_LEVEL_LOGNORM = 0;
int NORMAL_ABS = 0;
int CONST_FATAL = 0;

//      Mean,      normSD,      Skew,      Kurtosis,      r^2
//Ideal  80.0171    0.1722    -1.0700    4.8656    1.0000
//Sim    80.3812    0.2230    -1.3577    6.8501    0.9453
// Population = 8000
double BASAL_HOMOGENEITY = 0.45000000;
double BASAL_RATE_SD = 0.20000000;
double BASAL_RATE = 0.00002560;
double EXP_RATE = 0.04242741;
double EXP_RATE_SD = 0.22222222;
double EXP_HOMOGENEITY = 0.79101563;
double FATAL_LEVEL = 0.90000000;
double FATAL_LEVEL_SD = 0.31978680;
/**/

/* normal reflected, low FATAL_SD - * /
int MECH_PER_CATEGORY = 5;
double MECH_DIVISOR[] = {1,2,4,8,8};
//double MECH_DIVISOR[] = {2.5,2.5,2.5,2.5,2.5};
double CAT_MULT[NUM_CATEGORIES] = {1,1,1,1,1,1,1};
int FATAL_LEVEL_LOGNORM = 0;
int NORMAL_ABS = 1;
int CONST_FATAL = 0;

double BASAL_HOMOGENEITY = 0.45000000;
double BASAL_RATE_SD = 0.20000000;
double BASAL_RATE = 0.00005000;
double EXP_RATE = 0.03629930;
double EXP_RATE_SD = 0.22222222;
double EXP_HOMOGENEITY = 0.50625000;
double FATAL_LEVEL = 1.12500000;
double FATAL_LEVEL_SD = 0.37974683;
/**/

```

```

/* normal reflected - good fit* /
int MECH_PER_CATEGORY = 5;
double MECH_DIVISOR[] = {1,2,4,8,8};
//double MECH_DIVISOR[] = {2.5,2.5,2.5,2.5,2.5};
double CAT_MULT[NUM_CATEGORIES] = {1,1,1,1,1,1,1};
int FATAL_LEVEL_LOGNORM = 0;
int NORMAL_ABS = 1;

double BASAL_HOMOGENEITY = 0.45000000;
double BASAL_RATE_SD = 0.20000000;
double BASAL_RATE = 0.00005000;
double EXP_RATE = 0.03629930;
double EXP_RATE_SD = 0.22222222;
double EXP_HOMOGENEITY = 0.50625000;
double FATAL_LEVEL = 1.12500000;
double FATAL_LEVEL_SD = 0.37974683;
//double BASAL_HOMOGENEITY = 0.45000000;
//double BASAL_RATE_SD = 0.20000000;
//double BASAL_RATE = 0.00005000;
//double EXP_RATE = 0.03722409;
//double EXP_RATE_SD = 0.22222222;
//double EXP_HOMOGENEITY = 0.40500000;
//double FATAL_LEVEL = 1.25000000;
//double FATAL_LEVEL_SD = 0.39473684;
/**/

/* normal reflected, zero SD - OK fit to Gompertz* /
int MECH_PER_CATEGORY = 5;
double MECH_DIVISOR[] = {1,2,4,8,8};
//double MECH_DIVISOR[] = {2.5,2.5,2.5,2.5,2.5};
double CAT_MULT[NUM_CATEGORIES] = {1,1,1,1,1,1,1};
int FATAL_LEVEL_LOGNORM = 0;
int NORMAL_ABS = 1;
int CONST_FATAL = 0;
//      Mean,      normSD,      Skew,      Kurtosis,      r^2
//Ideal  80.0171    0.1722    -1.0700    4.8656    1.0000
//Sim    79.8992    0.1918    -1.1908    4.8085    0.9650
// Population = 16000

double BASAL_HOMOGENEITY = 0.45000000;
double BASAL_RATE_SD = 0.00000051;
double BASAL_RATE = 0.00005000;

```

```

double EXP_RATE = 0.03882040;
double EXP_RATE_SD = 0.00000022;
double EXP_HOMOGENEITY = 0.50625000;
double FATAL_LEVEL = 3.05175781;
double FATAL_LEVEL_SD = 0.50441802;
/**/

/* normal reflected - good fit* /
int MECH_PER_CATEGORY = 5;
double MECH_DIVISOR[] = {1,2,4,8,16};
double CAT_MULT[NUM_CATEGORIES] = {1,1,1,1,1,1,1};
int FATAL_LEVEL_LOGNORM = 0;
int NORMAL_ABS = 1;
int CONST_FATAL = 0;

double BASAL_HOMOGENEITY = 0.45000000;
double BASAL_RATE_SD = 0.20000000;
double BASAL_RATE = 0.00005000;
double EXP_RATE = 0.03722409;
double EXP_RATE_SD = 0.22222222;
double EXP_HOMOGENEITY = 0.40500000;
double FATAL_LEVEL = 1.25000000;
double FATAL_LEVEL_SD = 0.39473684;
/**/
//Cure 1/2 each time
//1,2,4,8,16,32,64,128,256,512,1024,2048,4096,8192,16384,32768,65536,131072,262144,524288,
1048576,2097152,4194304,8388608,16777216,33554432,67108864,134217728,268435456,268435456
//Cure 1/3 each time
//1.5,2.25,3.38,5.06,7.59,11.39,17.09,25.63,38.44,57.67,86.5,129.75,194.62,291.93,437.89,6
56.84,985.26,1477.89,2216.84,3325.26,4987.89,7481.83,11222.74,16834.11,25251.17,37876.75,5
6815.13,85222.69,127834.04,63917.02]
//Cure 1/2, 1/3, 1/4...
//1,3,6,10,15,21,28,36,45,55,66,78,91,105,120,136,153,171,190,210,231,253,276,300,325,351,
378,406,435,15
//Cure 1/4, 1/5, 1/6...
//2,3.33,5,7,9.33,12,15,18.33,22,26,30.33,35,40,45.33,51,57,63.33,70,77,84.33,92,100,108.3
3,117,126,135.33,145,155,165.33,5.33,
//Cure 1/2, 1/2.5, 1/3, 1/3.5...
//1,2.5,5,8.75,14,21,30,41.25,55,71.5,91,113.75,140,170,204,242.25,285,332.5,385,442.75,50
6,575,650,731.25,819,913.5,1015,1123.75,1240,82.67

/* normal reflected - good fit* / // Cure 1/2 each time
int MECH_PER_CATEGORY = 30;
double MECH_DIVISOR[] = {1,2,4,8,16,32,64,128,256,512,1024,2048,4096,8192,
16384,32768,65536,131072,262144,524288,1048576,2097152,4194304,8388608,
16777216.0,33554432.0,67108864.0,134217728.0,268435456.0,2684354568192.0};
double CAT_MULT[NUM_CATEGORIES] = {1,1,1,1,1,1,1};

```

```

int FATAL_LEVEL_LOGNORM = 0;
int NORMAL_ABS = 1;

//      Mean,      normSD,      Skew,      Kurtosis,      r^2
//Ideal  80.0171    0.1722    -1.0700    4.8656    1.0000
//Sim    80.7728    0.1706    -1.0902    5.1991    0.9841
// Population = 8000

double BASAL_HOMOGENEITY = 0.45000000;
double BASAL_RATE_SD = 0.20000000;
double BASAL_RATE = 0.00004858;
double EXP_RATE = 0.03770721;
double EXP_RATE_SD = 0.30864198;
double EXP_HOMOGENEITY = 0.32000000;
double FATAL_LEVEL = 1.46198830;
double FATAL_LEVEL_SD = 0.37980769;
/**/

/* normal reflected - good fit* /    //Cure 1/2, 1/2.5, 1/3, 1/3.5...
int MECH_PER_CATEGORY = 30;
double MECH_DIVISOR[] = {1,2.5,5,8.75,14,21,30,41.25,55,71.5,91,113.75,140,170,
                        204,242.25,285,332.5,385,442.75,506,575,650,731.25,819,
                        913.5,1015,1123.75,1240,82.67};
double CAT_MULT[NUM_CATEGORIES] = {1,1,1,1,1,1,1};
int FATAL_LEVEL_LOGNORM = 0;
int NORMAL_ABS = 1;

//      Mean,      normSD,      Skew,      Kurtosis,      r^2
//Ideal  80.0171    0.1722    -1.0700    4.8656    1.0000
//Sim    80.8252    0.1701    -1.0737    5.1814    0.9871
// Population = 8000

double BASAL_HOMOGENEITY = 0.45000000;
double BASAL_RATE_SD = 0.20000000;
double BASAL_RATE = 0.00004858;
double EXP_RATE = 0.03770721;
double EXP_RATE_SD = 0.30864198;
double EXP_HOMOGENEITY = 0.32000000;
double FATAL_LEVEL = 1.46198830;
double FATAL_LEVEL_SD = 0.37980769;
/**/

/* normal reflected - good fit* /    // Cure 1/3 each time

```



```

int MECH_PER_CATEGORY = 30;
double MECH_DIVISOR[] =
{1.5,2.25,3.38,5.06,7.59,11.39,17.09,25.63,38.44,57.67,86.5,129.75,
  194.62,291.93,437.89,656.84,985.26,1477.89,2216.84,3325.26,4987.89,
  7481.83,11222.74,16834.11,25251.17,37876.75,56815.13,85222.69,
  127834.04,63917.02};
double CAT_MULT[NUM_CATEGORIES] = {1,1,1,1,1,1,1};
int FATAL_LEVEL_LOGNORM = 0;
int NORMAL_ABS = 1;

//      Mean,      normSD,      Skew,      Kurtosis,      r^2
//Ideal  80.0171   0.1722   -1.0700   4.8656   1.0000
//Sim    81.2735   0.1696   -1.0879   5.0940   0.9839
// Population = 8000

double BASAL_HOMOGENEITY = 0.45000000;
double BASAL_RATE_SD = 0.20000000;
double BASAL_RATE = 0.00004858;
double EXP_RATE = 0.03770721;
double EXP_RATE_SD = 0.30864198;
double EXP_HOMOGENEITY = 0.32000000;
double FATAL_LEVEL = 1.46198830;
double FATAL_LEVEL_SD = 0.37980769;
/**/
/* normal reflected - good fit* / // Cure 1/2, 1/3, 1/4...
int MECH_PER_CATEGORY = 30;
double MECH_DIVISOR[] = {1,3,6,10,15,21,28,36,45,55,66,78,91,105,120,136,153,171,190,210,
  231,253,276,300,325,351,378,406,435,15};
double CAT_MULT[NUM_CATEGORIES] = {1,1,1,1,1,1,1};
int FATAL_LEVEL_LOGNORM = 0;
int NORMAL_ABS = 1;

#if 0
//from earlier 1/N
//      Mean,      normSD,      Skew,      Kurtosis,      r^2
//Ideal  80.0171   0.1722   -1.0700   4.8656   1.0000
//Sim    80.0860   0.1766   -1.0961   5.1927   0.9819
// Population = 8000
double BASAL_HOMOGENEITY = 0.45000000;
double BASAL_RATE_SD = 0.20000000;
double BASAL_RATE = 0.00005263;
double EXP_RATE = 0.03675318;
double EXP_RATE_SD = 0.27434843;
double EXP_HOMOGENEITY = 0.40000000;
double FATAL_LEVEL = 1.31578947;
double FATAL_LEVEL_SD = 0.38392380;

```

```

#else
//from earlier 1/2
//      Mean,      normSD,      Skew,      Kurtosis,      r^2
//Ideal  80.0171    0.1722    -1.0700    4.8656    1.0000
//Sim    81.0839    0.1699    -1.0875    5.0973    0.9870
// Population = 8000
double BASAL_HOMOGENEITY = 0.45000000;
double BASAL_RATE_SD = 0.20000000;
double BASAL_RATE = 0.00004858;
double EXP_RATE = 0.03770721;
double EXP_RATE_SD = 0.30864198;
double EXP_HOMOGENEITY = 0.32000000;
double FATAL_LEVEL = 1.46198830;
double FATAL_LEVEL_SD = 0.37980769;
#endif
/**/
/* normal reflected - good fit* / // Cure 1/4, 1/5, 1/6...
int MECH_PER_CATEGORY = 30;
double MECH_DIVISOR[] = {2,3.33,5,7,9.33,12,15,18.33,22,26,30.33,35,40,45.33,51,57,63.33,
                        70,77,84.33,92,100,108.33,117,126,135.33,145,155,165.33,5.33};
double CAT_MULT[NUM_CATEGORIES] = {1,1,1,1,1,1,1};
int FATAL_LEVEL_LOGNORM = 0;
int NORMAL_ABS = 1;

//      Mean,      normSD,      Skew,      Kurtosis,      r^2
//Ideal  80.0171    0.1722    -1.0700    4.8656    1.0000
//Sim    81.7828    0.1645    -1.1589    5.4316    0.9807
// Population = 8000
double BASAL_HOMOGENEITY = 0.45000000;
double BASAL_RATE_SD = 0.20000000;
double BASAL_RATE = 0.00004858;
double EXP_RATE = 0.03770721;
double EXP_RATE_SD = 0.30864198;
double EXP_HOMOGENEITY = 0.32000000;
double FATAL_LEVEL = 1.46198830;
double FATAL_LEVEL_SD = 0.37980769;
/**/

/* normal reflected - good fit* / ////Cure 1/4 each time
int MECH_PER_CATEGORY = 15;
double MECH_DIVISOR[] =
{1,1.33,1.78,2.37,3.16,4.21,5.62,7.49,9.99,13.32,17.76,23.68,31.57,42.09,14.03};
double CAT_MULT[NUM_CATEGORIES] = {1.2,1.1,1,0.9,0.8,0.7,0.6};
int FATAL_LEVEL_LOGNORM = 0;

```

```

int NORMAL_ABS = 1;
//      Mean,      normSD,      Skew,      Kurtosis,      r^2
//Ideal  80.0171   0.1722   -1.0700   4.8656   1.0000
//Sim    80.3307   0.1755   -1.1611   5.2429   0.9913
// Population = 16000

double BASAL_HOMOGENEITY = 0.45000000;
double BASAL_RATE_SD = 0.20000000;
double BASAL_RATE = 0.00003947;
double EXP_RATE = 0.02011371;
double EXP_RATE_SD = 0.20071215;
double EXP_HOMOGENEITY = 0.50000000;
double FATAL_LEVEL = 2.05592105;
double FATAL_LEVEL_SD = 0.38367138;
/**/

/* normal reflected - good fit* / // Cure 1/2 each time
int MECH_PER_CATEGORY = 15;
double MECH_DIVISOR[] = {1,2,4,8,16,32,64,128,256,512,1024,2048,4096,8192,8192};
double CAT_MULT[NUM_CATEGORIES] = {1,1,1,1,1,1,1};
int FATAL_LEVEL_LOGNORM = 0;
int NORMAL_ABS = 1;

//      Mean,      normSD,      Skew,      Kurtosis,      r^2
//Ideal  80.0171   0.1722   -1.0700   4.8656   1.0000
//Sim    80.9397   0.1684   -1.0840   5.3031   0.9893
// Population = 16000

double BASAL_HOMOGENEITY = 0.45000000;
double BASAL_RATE_SD = 0.20000000;
double BASAL_RATE = 0.00004858;
double EXP_RATE = 0.03770721;
double EXP_RATE_SD = 0.30864198;
double EXP_HOMOGENEITY = 0.32000000;
double FATAL_LEVEL = 1.46198830;
double FATAL_LEVEL_SD = 0.37980769;
/**/

/* normal reflected - good fit* / ///Cure 1/4 each time
int MECH_PER_CATEGORY = 15;
double MECH_DIVISOR[] =
{1,1.33,1.78,2.37,3.16,4.21,5.62,7.49,9.99,13.32,17.76,23.68,31.57,42.09,14.03};
double CAT_MULT[NUM_CATEGORIES] = {1,1,1,1,1,1,1};

```

```

int FATAL_LEVEL_LOGNORM = 0;
int NORMAL_ABS = 1;
//      Mean,      normSD,      Skew,      Kurtosis,      r^2
//Ideal  80.0171    0.1722     -1.0700    4.8656    1.0000
//Sim    80.7430    0.1759     -1.0144    4.9862    0.9844
// Population = 8000

double BASAL_HOMOGENEITY = 0.45000000;
double BASAL_RATE_SD = 0.20000000;
double BASAL_RATE = 0.00003499;
double EXP_RATE = 0.01930899;
double EXP_RATE_SD = 0.20252488;
double EXP_HOMOGENEITY = 0.55555556;
double FATAL_LEVEL = 2.27802886;
double FATAL_LEVEL_SD = 0.39376800;
/**/

/* normal reflected - good fit* / ////Cure 1/4 each time
int MECH_PER_CATEGORY = 15;
double MECH_DIVISOR[] =
{1,1.33,1.78,2.37,3.16,4.21,5.62,7.49,9.99,13.32,17.76,23.68,31.57,42.09,14.03};
double CAT_MULT[NUM_CATEGORIES] = {1,1,1,1,1,1,1};
int FATAL_LEVEL_LOGNORM = 0;
int NORMAL_ABS = 1;
//      Mean,      normSD,      Skew,      Kurtosis,      r^2
//Ideal  80.0171    0.1722     -1.0700    4.8656    1.0000
//Sim    81.1780    0.1665     -1.0472    5.3162    0.9793
// Population = 8000

double BASAL_HOMOGENEITY = 0.45000000;
double BASAL_RATE_SD = 0.20000000;
double BASAL_RATE = 0.00003888;
double EXP_RATE = 0.01626020;
double EXP_RATE_SD = 0.21318408;
double EXP_HOMOGENEITY = 0.55555556;
double FATAL_LEVEL = 0.93308062;
double FATAL_LEVEL_SD = 0.34649123;/**/

/* normal reflected - good fit* / ////Cure 1/4 each time
int MECH_PER_CATEGORY = 15;
double MECH_DIVISOR[] =
{1,1.33,1.78,2.37,3.16,4.21,5.62,7.49,9.99,13.32,17.76,23.68,31.57,42.09,14.03};
double CAT_MULT[NUM_CATEGORIES] = {1,1,1,1,1,1,1};
int FATAL_LEVEL_LOGNORM = 0;
int NORMAL_ABS = 1;

```

```
//      Mean,      normSD,      Skew,      Kurtosis,      r^2
//Ideal  80.0171    0.1722    -1.0700    4.8656    1.0000
//Sim    79.9961    0.1851    -1.1368    5.0180    0.9913
// Population = 8000

double BASAL_HOMOGENEITY = 0.45000000;
double BASAL_RATE_SD = 0.20000000;
double BASAL_RATE = 0.00003158;
double EXP_RATE = 0.01741491;
double EXP_RATE_SD = 0.18289895;
double EXP_HOMOGENEITY = 0.50000000;
double FATAL_LEVEL = 1.31578947;
double FATAL_LEVEL_SD = 0.39376800;
/**/

/* normal reflected - good fit* / //Cure 1/N each N'th time
int MECH_PER_CATEGORY = 15;
double MECH_DIVISOR[] = {1,3,6,10,15,21,28,36,45,55,66,78,91,105,7.5};
double CAT_MULT[NUM_CATEGORIES] = {1,1,1,1,1,1,1,1};
int FATAL_LEVEL_LOGNORM = 0;
int NORMAL_ABS = 1;

double BASAL_HOMOGENEITY = 0.45000000;
double BASAL_RATE_SD = 0.20000000;
double BASAL_RATE = 0.00005263;
double EXP_RATE = 0.03675318;
double EXP_RATE_SD = 0.27434843;
double EXP_HOMOGENEITY = 0.40000000;
double FATAL_LEVEL = 1.31578947;
double FATAL_LEVEL_SD = 0.38392380;
/**/

/* normal reflected - good fit* /
int MECH_PER_CATEGORY = 15;
double MECH_DIVISOR[] = {1,1,1,1,1,1,1,1,1,1,1,1,1,1,1};
double CAT_MULT[NUM_CATEGORIES] = {1,1,1,1,1,1,1,1};
int FATAL_LEVEL_LOGNORM = 0;
int NORMAL_ABS = 1;

double BASAL_HOMOGENEITY = 0.45000000;
double BASAL_RATE_SD = 0.20000000;
double BASAL_RATE = 0.00005263;
double EXP_RATE = 0.00293211;
```

```

double EXP_RATE_SD = 0.30864198;
double EXP_HOMOGENEITY = 0.40000000;
double FATAL_LEVEL = 1.31578947;
double FATAL_LEVEL_SD = 0.32825485;
/**/

/* normal reflected - good fit* /
int MECH_PER_CATEGORY = 15;
double MECH_DIVISOR[] = {1,2,3,4,5,6,7,8,9,10,11,12,13,14,15};
double CAT_MULT[NUM_CATEGORIES] = {1,1,1,1,1,1,1};
int FATAL_LEVEL_LOGNORM = 0;
int NORMAL_ABS = 1;

double BASAL_HOMOGENEITY = 0.45000000;
double BASAL_RATE_SD = 0.20000000;
double BASAL_RATE = 0.00005263;
double EXP_RATE = 0.01954742;
double EXP_RATE_SD = 0.24691358;
double EXP_HOMOGENEITY = 0.40000000;
double FATAL_LEVEL = 1.31578947;
double FATAL_LEVEL_SD = 0.38980263;
/**/

/* normal reflected - good fit* /
int MECH_PER_CATEGORY = 5;
double MECH_DIVISOR[] = {1,2,3,4,5};
double CAT_MULT[NUM_CATEGORIES] = {1,1,1,1,1,1,1};
int FATAL_LEVEL_LOGNORM = 0;
int NORMAL_ABS = 1;
double BASAL_HOMOGENEITY = 0.45000000;
double BASAL_RATE_SD = 0.20000000;
double BASAL_RATE = 0.00005263;
double EXP_RATE = 0.03054284;
double EXP_RATE_SD = 0.24691358;
double EXP_HOMOGENEITY = 0.40000000;
double FATAL_LEVEL = 1.31578947;
double FATAL_LEVEL_SD = 0.38980263;
/**/

/* normal reflected - good fit* /
int MECH_PER_CATEGORY = 5;
double MECH_DIVISOR[] = {2,4,6,8,10};
double CAT_MULT[NUM_CATEGORIES] = {1.2,1.1,1,0.9,0.8,0.7,0.6};
int FATAL_LEVEL_LOGNORM = 0;

```

```

int NORMAL_ABS = 1;
double BASAL_HOMOGENEITY = 0.45000000;
double BASAL_RATE_SD = 0.20000000;
double BASAL_RATE = 0.00004678;
double EXP_RATE = 0.08350602;
double EXP_RATE_SD = 0.30483158;
double EXP_HOMOGENEITY = 0.25600000;
double FATAL_LEVEL = 2.03053931;
double FATAL_LEVEL_SD = 0.42756476;
/**/

/* normal reflected - good fit* /
int MECH_PER_CATEGORY = 5;
double MECH_DIVISOR[] = {2,4,6,8,10};
double CAT_MULT[NUM_CATEGORIES] = {1,1,1,1,1,1,1};
int FATAL_LEVEL_LOGNORM = 0;
int NORMAL_ABS = 1;

double BASAL_HOMOGENEITY = 0.45000000;
double BASAL_RATE_SD = 0.20000000;
double BASAL_RATE = 0.00005848;
double EXP_RATE = 0.07070102;
double EXP_RATE_SD = 0.27434842;
double EXP_HOMOGENEITY = 0.40000000;
double FATAL_LEVEL = 1.46198830;
double FATAL_LEVEL_SD = 0.40006059;
/**/

/* High basal; poor with or without refl norm, impossible with lognorm* /
int MECH_PER_CATEGORY = 5;
double MECH_DIVISOR[] = {1,2,4,8,16};
double CAT_MULT[NUM_CATEGORIES] = {1,1,1,1,1,1,1};
int FATAL_LEVEL_LOGNORM = 0;
int NORMAL_ABS = 1;

double BASAL_HOMOGENEITY = 0.45000000;
double BASAL_RATE_SD = 0.20000000;
double BASAL_RATE = 0.01323489;
double EXP_RATE = 0.00319752;
double EXP_RATE_SD = 0.22222222;
double EXP_HOMOGENEITY = 0.40500000;
double FATAL_LEVEL = 5.78703704;
double FATAL_LEVEL_SD = 0.25263158;
/**/

```

```

/* lognorm - impossible*/

/* normal unreflected - poor fit* /
int MECH_PER_CATEGORY = 5;
double MECH_DIVISOR[] = {1,2,3,4,5};
double CAT_MULT[NUM_CATEGORIES] = {1,1,1,1,1,1,1};
int FATAL_LEVEL_LOGNORM = 0;
int NORMAL_ABS = 0;

double BASAL_HOMOGENEITY = 0.45000000;
double BASAL_RATE_SD = 0.20000000;
double BASAL_RATE = 0.00005263;
double EXP_RATE = 0.03666269;
double EXP_RATE_SD = 0.24691358;
double EXP_HOMOGENEITY = 0.78125000;
double FATAL_LEVEL = 1.31578947;
double FATAL_LEVEL_SD = 0.29625000;
/**/

/* Exp divisor: Lognormal: Can't be done!*/

/* Numbers for normal unreflected - semi-good fit * /
int MECH_PER_CATEGORY = 5;
double MECH_DIVISOR[] = {1,2,4,8,16};
double CAT_MULT[NUM_CATEGORIES] = {1,1,1,1,1,1,1};
int FATAL_LEVEL_LOGNORM = 0;
int NORMAL_ABS = 0;

double BASAL_HOMOGENEITY = 0.45000000;
double BASAL_RATE_SD = 0.20000000;
double BASAL_RATE = 0.00013563;
double EXP_RATE = 0.02821194;
double EXP_RATE_SD = 0.43945313;
double EXP_HOMOGENEITY = 0.47460938;
double FATAL_LEVEL = 0.64000000;
double FATAL_LEVEL_SD = 0.28936406;
/**/

```



```

/* OLD stuff - from Python?
#define NUM_CATEGORIES 7
#define MECH_PER_CATEGORY 5
#define MAX_AGE 1000
#define POPULATION 5000

double MECH_DIVISOR = [1,2,4,8,16];
double BASAL_RATE = 0.000050 , BASAL_RATE_SD = 0.2 , BASAL_RATE_LOGNORM = 1;
double BASAL_HOMOGENEITY = 0.45
double EXP_RATE = 2**((1.0/20)-1) , EXP_RATE_SD = 0.2 , EXP_RATE_LOGNORM = 1;
double EXP_HOMOGENEITY = 0.45;
double FATAL_LEVEL = 1.0 , FATAL_LEVEL_SD = 0.4 , FATAL_LEVEL_LOGNORM = 0;
*/

```

stats.cpp

```

#include "LEScon.h"

#define GMODEL_SPAN 130 /* Max age to run the model */
#define GMODEL_MAX 112.5 /* "max age" for significant p(alive) */
#define GMODEL_DUB 7.5 /* Doubling period for p(death) */
#define AMODEL_SPAN 113
double gmodel[GMODEL_SPAN];
double amodel[AMODEL_SPAN];
double AM_MORT[AMODEL_SPAN] =
{0.00636,0.00047,0.00030,0.00024,0.00019,0.00017,0.00016,0.00016,
0.00014,0.00014,0.00012,0.00012,0.00015,0.00018,0.00025,0.00033,0.00038,0.00044,0.00045,
0.00046,0.00047,0.00048,0.00048,0.00049,0.00051,0.00052,0.00053,0.00056,0.00058,0.00062,
0.00065,0.00070,0.00075,0.00082,0.00089,0.00098,0.00106,0.00116,0.00124,0.00134,0.00145,
0.00156,0.00169,0.00182,0.00196,0.00213,0.00231,0.00251,0.00272,0.00296,0.00322,0.00351,
0.00384,0.00423,0.00465,0.00513,0.00565,0.00621,0.00679,0.00741,0.00807,0.00884,0.00973,
0.01076,0.01193,0.01324,0.01464,0.01602,0.01735,0.01869,0.02023,0.02200,0.02395,0.02607,
0.02841,0.03114,0.03430,0.03785,0.04179,0.04623,0.05133,0.05711,0.06356,0.07061,0.07848,
0.08721,0.09696,0.10778,0.11973,0.13285,0.14712,0.16246,0.17891,0.19642,0.21489,0.23376,
0.25264,0.27113,0.28932,0.30679,0.32489,0.34445,0.36514,0.38677,0.41176,0.43333,0.46078,
0.49091,0.50000,0.57143,0.50000,0.66667,1.00000};

STATS amStats, gmStats;

static int modelsFilled = 0;

void fillModels(void)
{
    double b=log(2)/GMODEL_DUB;
    double A=1/exp(GMODEL_MAX*b);
    double pop = 1;

```

```

    for (int age = 0; age < GMODEL_SPAN; age++) {
        gmodel[age] = pop* (1 - exp((A *exp(b *age)) * (1 - exp(b)) /b ));
        pop -= gmodel[age];
    }
    pop = 1;
    for (int age = 0; age < AMODEL_SPAN; age++) {
        amodel[age] = pop*AM_MORT[age];
        pop -= amodel[age];
    }
    modelsFilled = 1;
    statistics(amodel, AMODEL_SPAN, amStats);
    statistics(gmodel, GMODEL_SPAN, gmStats);
}

int statStart = 0;
double rSquared(double x[], double y[], int n)
{
    double mx=0, my=0, sx2=0, sy2=0, sxy=0;
    for (int i=statStart; i<n; i++) {
        mx += x[i];
        my += y[i];
        sx2 += x[i]*x[i];
        sy2 += y[i]*y[i];
        sxy += x[i]*y[i];
    }
    mx /= n; my /= n;
    double ssxy = sxy-n*mx*my;
    double ssxx = sx2-n*mx*mx;
    double ssyy = sy2-n*my*my;
    return ssxy*ssxy/(ssxx*ssyy);
}

void statistics(double dist[], int len, STATS &s)
{
    if (!modelsFilled) {
        fillModels();
    }
    double mean, sd, skew, kurt;
    double Sum = 0.0;
    double n = 0.0;
    for (int i=statStart; i < len; i++) {
        Sum += i*dist[i];
    }

```

```

        n += dist[i];
    }
    mean = Sum/n;
    double variance = 0;
    skew = 0;
    kurt = 0;
    double deviation;
    for (int i=statStart; i < len; i++) {
        deviation = i - mean;
        variance += dist[i]*pow(deviation,2);
        skew += dist[i]*pow(deviation,3);
        kurt += dist[i]*pow(deviation,4);
    }
    variance /= n;
    sd = sqrt(variance);
    if (variance > 0.0) {
        skew /= (n * variance * sd);
        kurt = kurt/(n * variance * variance);
    } else {
        skew = 0.0;
        kurt = 0.0;
    }
    s.mean = mean;
    s.sd = sd;
    s.nsd = sd/mean;
    s.skew = skew;
    s.kurt = kurt;
    s.ar2 = rSquared(amodel, dist, (len<AMODEL_SPAN)?len:AMODEL_SPAN);
    s.gr2 = rSquared(gmodel, dist, (len<GMODEL_SPAN)?len:GMODEL_SPAN);
}

```

challenge.cpp

```

#include "LEScon.h"

int WAIT_CHALLENGE = 0;

#define ARLEN(a) (sizeof(a)/sizeof(a[0]))

double cDamage [MAX_C_POP][MAX_C_AGE];
double pImDead [MAX_C_POP][MAX_C_AGE];
double cumPImDead [MAX_C_POP][MAX_C_AGE];
double totalDead [MAX_C_AGE];
double normTotalDead [MAX_C_AGE];

```

```

#define BUCKETS_PER_D_UNIT 400
#define MAX_D_UNIT 2
#define N_P_BUCKETS (BUCKETS_PER_D_UNIT * MAX_D_UNIT)
double pDeath[N_P_BUCKETS];
double cumPDeath[N_P_BUCKETS];

void initPDeath(void)
{
#define N_TRIALS 10000000
    ZERO(pDeath);
    for (int i = 0; i < N_TRIALS; i++) {
        double fatalLevel = dist(FATAL_LEVEL, FATAL_LEVEL_SD, FATAL_LEVEL_LOGNORM);
        int bnum = (int)(fatalLevel*BUCKETS_PER_D_UNIT);
        if (bnum < N_P_BUCKETS) {
            bnum = max(bnum, 0);
            pDeath[bnum] += 1.0/N_TRIALS;
        }
    }
    for (int b = 0; b < N_P_BUCKETS; b++) {
        printf("%f ", pDeath[b]);
    }
    printf("\n");
    for (int b = 1; b < N_P_BUCKETS; b++) {
        printf("%f ", pDeath[b]/pDeath[b-1]);
    }
    printf("\n");
}

void computeActuarial(void)
{
    ZERO(totalDead);
    cumPDeath[0] = pDeath[0];
    for (int b = 1; b < N_P_BUCKETS; b++) {
        cumPDeath[b] = cumPDeath[b-1] + pDeath[b];
        printf("%f ", cumPDeath[b]/cumPDeath[b-1]);
    }
    printf("\n");
    for (int p = 0; p < POPULATION; p++) {
        for (int a = 0; a < MAX_C_AGE; a++) {
            int damageBucket = (int)(cDamage[p][a]*BUCKETS_PER_D_UNIT);
            damageBucket = min(damageBucket, N_P_BUCKETS-1);
            double pAlreadyDead;

```

```

        if (a == 0) {
            pAlreadyDead = 0;
        } else {
            pAlreadyDead = cumPImDead[p][a-1];
        }
        pImDead[p][a] = cumPDeath[damageBucket] * (1-pAlreadyDead);
        cumPImDead[p][a] = pImDead[p][a] + pAlreadyDead;
        totalDead[a] += pImDead[p][a];
    }
}
for (int a = 0; a < MAX_C_AGE; a++) {
    normTotalDead[a] = totalDead[a] / POPULATION;
}
}

#define GH 200
#define CX 300
#define CY 300
#define DX 10
#define DY 510
#define LX 160
#define LY 510
#define GOMPX 310
#define GOMPY 510

int differ(double top, double bottom)
{
    return min(255, (int)(10000*(1-top/bottom)));
}

int cColor(double simD, double modelD)
{
    int red, green, blue;
    if (simD < modelD) {
        red = differ(simD, modelD); green = 0; blue = 255-red;
    } else {
        red = 0; green = differ(modelD, simD); blue = 255-green;
    }
    return COLOR(red,green,blue);
}

void plotCGomp(double array[], int len, int x, int y)
{
    double sum=array[0];
    for (int i=1; i<len; i++) {

```

```

        double gval = log(array[i]/(1-sum));
        putpixel(i+x, (int)(y-150-gval*15), cColor(array[i], gmodel[i]));
        sum += array[i];
    }
}

void plotCMortality(void)
{
    setColor(COLOR(255,255,255));
    drawGrid(CX,CY);
    setColor(COLOR(128,255,0));
    for (int i = 1; i < AMODEL_SPAN; i++) {
        line(i+CX, (int)(CY-amodel[i]*GH*30), i-1+CX, (int)(CY-amodel[i-1]*GH*30));
    }
    setColor(COLOR(255,128,0));
    for (int i = 1; i < GMODEL_SPAN; i++) {
        line(i+CX, (int)(CY-gmodel[i]*GH*30), i-1+CX, (int)(CY-gmodel[i-1]*GH*30));
    }
    for (int i = 0; i < MAX_C_AGE; i++) {
        putpixel (i+CX, (int)(CY-normTotalDead[i]*GH*30), cColor(normTotalDead[i],
gmodel[i]));
    }
    setColor(COLOR(128,128,0));
    line(GOMPX, GOMPY, GOMPX, GOMPY-150);
    line(GOMPX, GOMPY, GOMPX+120, GOMPY);
    plotGomp(gmodel, GMODEL_SPAN, COLOR(128,255,0), GOMPX, GOMPY);
    plotGomp(amodel, AMODEL_SPAN, COLOR(255,128,0), GOMPX, GOMPY);
    plotCGomp(normTotalDead, MAX_C_AGE, GOMPX, GOMPY);
}

#define P_SCALE 10000
int prPos = 10;
int curBuck = 50;
void plotDamage(void)
{
    setColor(COLOR(128,128,128));
    drawGrid(DX-1,DY-1);
    outtextxy(DX, DY+30, "Prob Distribution");
    for (int b = 0; b < N_P_BUCKETS; b++) {
        if (b == curBuck)
            setColor(COLOR(255,0,255));
        else
            setColor(COLOR(55,55,0));
        line(DX, DY-b, (int)(DX+pDeath[b]*P_SCALE), DY-b);
    }
}

```

```

    setcolor(COLOR(128,128,128));
    drawGrid(LX-1,LY-1);
    outtextxy(LX, LY+30, "Fatal Level");
    for (int age=0; age < MAX_C_AGE; age++) {
        for (int pop = 0; pop < POPULATION; pop++) {
            putpixel(LX+age, (int)(LY-cDamage[pop][age]*BUCKETS_PER_D_UNIT),
                    cColor(normTotalDead[age], gmodel[age]));
        }
    }
}

char Cscreen[800*600*4+4];
void writeScreenC(void)
{
    ofstream outfile ("c:/cscreen.raw", ios::trunc|ios::binary);
    getimage(0, 0, 799, 599, Cscreen);
    for (int y=0; y<600; y++) {
        for (int x=0;x<800;x++) {
            int c=getpixel(x,y);
            outfile.write(&Cscreen[y*3200+x*4],3);
        }
    }
    outfile.close();
}

void wait(void) {          char ch = getch(); }
Damage Cdamages[MAX_POP];
void playwithChallenge(void)
{
    bool actuarial = 1;
    initwindow(800,600);
    setttextstyle(9, HORIZ_DIR, 0);
    setusercharsize(1,3,1,3);

    WAIT_CHALLENGE = 1;
    POPULATION = 50;
    double normYearDeaths[MAX_AGE+1];
    setcolor(COLOR(255,255,0));
    outtextxy(0,10,"Filling probability... ");      kbhit();
    initPDeath();
    outtextxy(0,10,"Calculating damages... ");      kbhit();
    fillModels();

```

```

test(normYearDeaths, Cdamages);
outtextxy(0,10,"
");
int quit=0;
STATS s;
while(!quit) {
    prPos = 10;
    computeActuarial();
    cleardevice();
    plotCMortality();
    plotDamage();

    char outstr[200];
    setcolor(COLOR(255,255,0));
    sprintf(outstr, "Population = %d",POPULATION);
    outtextxy(0,prPos, outstr); prPos += 20;
    outtextxy(0,prPos, actuarial?"Actuarial":"Gompertz"); prPos += 20;
    sprintf(outstr, "Stat Range Start = %d",statStart);
    outtextxy(0,prPos, outstr); prPos += 20;
    sprintf(outstr, "Cum prob = %f",cumPDeath[N_P_BUCKETS-1]);
    outtextxy(0,prPos, outstr); prPos += 20;
    sprintf(outstr, "BASAL_RATE_SD = %f",BASAL_RATE_SD);
    outtextxy(0,prPos, outstr); prPos += 20;
    sprintf(outstr, "EXP_RATE_SD = %f",EXP_RATE_SD);
    outtextxy(0,prPos, outstr); prPos += 20;
    sprintf(outstr,"%23s    Mean,    normSD,    Skew, Kurtosis, ar^2,    gr^2"," ");
#define TX CX+40
    outtextxy(TX,20,outstr);
    sprintf(outstr,"%23s    %3.4f    %3.4f    %3.4f    %3.4f    %3.4f    %3.4f","Actuar",
        amStats.mean, amStats.nsd, amStats.skew, amStats.kurt,
        amStats.ar2, amStats.gr2);

    outtextxy(TX,40,outstr);
    sprintf(outstr,"%23s    %3.4f    %3.4f    %3.4f    %3.4f    %3.4f    %3.4f","Gomper",
        gmStats.mean, gmStats.nsd, gmStats.skew, gmStats.kurt,
        gmStats.ar2, gmStats.gr2);

    outtextxy(TX,60,outstr);
    statistics(normYearDeaths, MAX_C_AGE, s);
    sprintf(outstr,"%23s    %3.4f    %3.4f    %3.4f    %3.4f    %3.4f    %3.4f","Sim",
        s.mean,s.nsd,s.skew,s.kurt, s.ar2, s.gr2);
    outtextxy(TX,80,outstr);
    char ch = getch();
    setcolor(COLOR(255,255,255));
    outtextxy(TX+80,100,"WAIT...");
    switch(ch) {
        case 'P': if (POPULATION > 1) POPULATION /= 2; test(normYearDeaths,
Cdamages); continue;

```



```

        case 'p': POPULATION *= 2; test(normYearDeaths, Cdamages); continue;
        case 'w': writeScreenC(); continue;
        case 'b': BASAL_RATE_SD *= 1.5; test(normYearDeaths, Cdamages); continue;
        case 'B': BASAL_RATE_SD /= 1.5; test(normYearDeaths, Cdamages); continue;
        case 'v': EXP_RATE_SD *= 1.5; test(normYearDeaths, Cdamages); continue;
        case 'V': EXP_RATE_SD /= 1.5; test(normYearDeaths, Cdamages); continue;
        case 'q': quit=1; continue;
        case 'A': actuarial = 1; continue;
        case 'G': actuarial = 0; continue;
        case 'e': if (curBuck < N_P_BUCKETS-1) curBuck++; continue;
        case 'c': if (curBuck > 0) curBuck--; continue;
        case 'a': for (int b=curBuck; b<N_P_BUCKETS; b++) pDeath[b] *= 0.9;
continue;

        case 's': pDeath[curBuck] *= 0.9; continue;
        case 'd': pDeath[curBuck] *= 1.1111; continue;
        case 'f': for (int b=curBuck; b<N_P_BUCKETS; b++) pDeath[b] *= 1.1111;
continue;

        case 'r': if (statStart >= 20) statStart -= 20;
                    statistics(amodel, AMODEL_SPAN, amStats);
                    statistics(gmodel, GMODEL_SPAN, gmStats);
                    continue;
        case 'R': if (statStart <= 80) statStart += 20;
                    statistics(amodel, AMODEL_SPAN, amStats);
                    statistics(gmodel, GMODEL_SPAN, gmStats);
                    continue;
        case 'Z': for (int i = 0; i < N_P_BUCKETS; i++) pDeath[i] = 0.00001;
continue;
    }
}
closegraph();          //close graphics window
}

```

LEScon.dev

```

[Project]
FileName=LEScon.dev
Name=Project2
UnitCount=12
Type=1
Ver=1
ObjFiles=
Includes=
Libs=
PrivateResource=
ResourceIncludes=
MakeIncludes=

```

```
Compiler=
CppCompiler=
Linker=-lbgi -lgdi32_@@_
IsCpp=1
Icon=
ExeOutput=
ObjectOutput=
OverrideOutput=0
OverrideOutputName=LEScon.exe
HostApplication=
Folders=newran
CommandLine=
UseCustomMakefile=0
CustomMakefile=
IncludeVersionInfo=0
SupportXPThemes=0
CompilerSet=0
CompilerSettings=000000000000000000000000

[Unit1]
FileName=LEScon.cpp
CompileCpp=1
Folder=
Compile=1
Link=1
Priority=1000
OverrideBuildCmd=0
BuildCmd=

[VersionInfo]
Major=0
Minor=1
Release=1
Build=1
LanguageID=1033
CharsetID=1252
CompanyName=
FileVersion=
FileDescription=Developed using the Dev-C++ IDE
InternalName=
LegalCopyright=
LegalTrademarks=
OriginalFilename=
ProductName=
ProductVersion=
```

```
AutoIncBuildNr=0
[Unit2]
FileName=..\..\..\..\Dev-Cpp\include\winbgim.cpp
CompileCpp=1
Folder=
Compile=1
Link=1
Priority=1000
OverrideBuildCmd=0
BuildCmd=

[Unit3]
FileName=newran03\SIMPSTR.CPP
CompileCpp=1
Folder=newran
Compile=1
Link=1
Priority=1000
OverrideBuildCmd=0
BuildCmd=

[Unit4]
FileName=newran03\EXTREAL.CPP
CompileCpp=1
Folder=newran
Compile=1
Link=1
Priority=1000
OverrideBuildCmd=0
BuildCmd=

[Unit5]
FileName=newran03\MYEXCEPT.CPP
CompileCpp=1
Folder=newran
Compile=1
Link=1
Priority=1000
OverrideBuildCmd=0
BuildCmd=

[Unit6]
FileName=newran03\NEWRAN1.CPP
CompileCpp=1
```

```

Folder=newran
Compile=1
Link=1
Priority=1000
OverrideBuildCmd=0
BuildCmd=

[Unit7]
FileName=newran03\NEWLAN2.CPP
CompileCpp=1
Folder=newran
Compile=1
Link=1
Priority=1000
OverrideBuildCmd=0
BuildCmd=

[Unit9]
FileName=LEScon.h
CompileCpp=1
Folder=
Compile=1
Link=1
Priority=1000
OverrideBuildCmd=0
BuildCmd=

[Unit10]
FileName=params.cpp
CompileCpp=1
Folder=
Compile=1
Link=1
Priority=1000
OverrideBuildCmd=0
BuildCmd=

[Unit11]
FileName=stats.cpp
CompileCpp=1
Folder=
Compile=1
Link=1
Priority=1000
OverrideBuildCmd=0

```

```

BuildCmd=

[Unit8]
FileName=vary.cpp
CompileCpp=1
Folder=
Compile=1
Link=1
Priority=1000
OverrideBuildCmd=0
BuildCmd=

[Unit12]
FileName=challenge.cpp
CompileCpp=1
Folder=
Compile=1
Link=1
Priority=1000
OverrideBuildCmd=0
BuildCmd=

```

package available from <http://www.cs.colorado.edu/~main/cs1300/doc/bgi/bgi.html>, and Newran03, 28 September 2005, beta version, package available from <http://robertnz.net/download.html>.

Built with Bloodshed C++ compiler, available (free) from <http://www.bloodshed.net/devcpp.html>. See below for .dev file.

References

- de Grey ADNJ (2004) Escape velocity: why the prospect of extreme human life extension matters now. *PLoS Biol* 2:723–726
- de Grey ADNJ (2005) A strategy for postponing aging indefinitely. *Stud Health Technol Inform* 118:209–219
- de Grey ADNJ, Ames BN, Andersen JK, Bartke A, Campisi J, Heward CB, McCarter RJM, Stock G (2002) Time to talk SENS: critiquing the immutability of human aging. *Ann N Y Acad Sci* 959:452–462
- Fries JF (1980) Aging, natural death, and the compression of morbidity. *N Engl J Med* 303:130–135
- Gavrilov LA, Gavrilova NS (1991) The biology of life span: a quantitative approach. Harwood, New York
- Gavrilov LA, Gavrilova NS (2004) The reliability-engineering approach to the problem of biological aging. *Ann N Y Acad Sci* 1019:509–512

- Gompertz B (1825) On the nature of the function expressive of the law of human mortality, and on a new mode of determining the value of life contingencies. *Philos Trans R Soc London* 115:513–585
- Manton KG, Gu X, Lamb VL (2006) Change in chronic disability from 1982 to 2004/2005 as measured by long-term changes in function and health in the U.S. elderly population. *Proc Natl Acad Sci USA* 103:18374–18379
- Rockwood K, Mitnitski A (2006) Limits to deficit accumulation in elderly people. *Mech Ageing Dev* 127: 494–496
- Ukrainitseva SV, Yashin AI (2001) How individual age-associated changes may influence human morbidity and mortality patterns. *Mech Ageing Dev* 122:1447–1460
- Wilmoth JR (1997) Berkeley Mortality Database. Retrieved from <http://demog.berkeley.edu/~bmd> 19/9/2005