Casualty Actuarial Society
E-Forum, Summer 2018

The CAS *E-Forum*, Summer 2018

The Summer 2018 edition of the CAS *E-Forum* is a cooperative effort between the CAS *E-Forum* Committee and various CAS committees, task forces, working parties and special interest sections.

This *E-Forum* contains a report of the CAS Working Party on Sustainable ERM (SERM) and two independent research papers.

The CAS Working Party on Sustainable ERM (SERM)

Fan Yang, *Chairperson*

Sandra J. Callanan, FCAS
George E. Davis, FCAS
Voon Seng Lai, FCAS
Tom Wakefield

CAS *E-Forum*, Summer 2018

Table of Contents

For information on submitting a paper to the *E-Forum*, visit http://www.casact.org/pubs/forum/.

Introduction to Sustainable ERM

CAS Working Party on Sustainable ERM (SERM)

Abstract:

Motivation. With consumers and investors putting an increasing focus on Sustainability, traditional enterprise risk management (ERM) becomes less effective in describing an insurer's or reinsurer's true risk, true cost and true value due to lack of a framework to evaluate Environmental, Social and Governance (ESG) performance. This paper draws attention to the need to integrate Sustainability into daily business decisions of (re)insurance companies, thereby establishing a Sustainable ERM (SERM) framework. It provides definitions, methods and a framework to assist the transition to practicing SERM. A capital-based approach is employed to holistically capture human and natural capital indicators that may be left out in the traditional risk-based approach due to less precise measurements or the absence of a universally accepted causal relationship with the profit. This paper is intended to be an introductory paper. Further development and enhancement of the framework would benefit from input from actuaries in collaboration with other risk experts.

Method. The exploration of SERM started with extensive literature review on Sustainability and global trends. The financial logic of Sustainability programs is established to explore opportunities of embedding Sustainability into the ERM function. A preliminary SERM framework is developed from incorporating the industry's leading practices along with research done by thought leaders and institutions.

Results. In general, awareness, measurement and reporting in Sustainability need to be improved among the mainstream (re)insurers, particularly in the US. Industry leaders are promoting the ideas and shaping the best practices; however, it is evident that systematic consideration of Sustainability in general ERM is in its infancy. The contribution of this paper is twofold: to demonstrate the Sustainability imperative and to set a foundation for the SERM framework through establishing a common language and providing sample measurements and governance structure.

Conclusions. Sustainability is becoming a new norm in the corporate behavior, metrics and strategy of industry leaders. This paper serves as the first step towards its integration into ERM for (re)insurers. SERM enables methodology development and stewardship for comprehensive capital management encompassing financial, human and natural capital. If designed and implemented correctly, it improves stakeholder relationships and contributes to sustainable development of the firm as well as the society at large through holistic risk management.

Keywords. Sustainability, Sustainable Enterprise Risk Management (SERM), Enterprise Risk Management (ERM), financial capital, human capital, natural capital, Environmental, Social and Governance (ESG).

1 INTRODUCTION

The concept of Sustainability is evolving as society changes in response to the urgent need to move towards an environmentally, socially and economically sustainable future. At its core, Sustainability is an objective to create long-term business value through preservation and enhancement of financial, human and natural capital. This objective is increasingly established by individuals, corporations and non-corporate organizations of all types. At the national and global level, the objective of Sustainability is reflected in the policies of supporting a green and inclusive economy through realizing The United Nations Sustainable Development Goals (SDGs) [1]. In the private sector, Sustainability entails more than green initiatives or corporate social responsibility (CSR); it inspires long-term business value creation by simultaneously improving corporate performance in utilizing the financial, human and natural resources. These resources are the foundation for financial, human and natural capital that enables corporate value creation. The whole process integrates the

management of Environmental, Social and Governance (ESG) risks in the acquisition, development and deployment of multiple capitals, not just financial capital.

Insurers are not traditionally conditioned to consider business management in terms of multiple capitals or the management of ESG risks. However, there is an interdependency of the environment, society, and business. In fact, many issues an organization faces can be attributed to its failure to perceive the interactions and long-term implications on business outcome. Recognizing this, enlightened businesses started to engage in and explore an environmental and social domain that was previously not a top item of the agenda. The term 'Sustainability' as well as associated practices emerged in various aspects, ranging from product development, branding, corporate governance, human capital management, to community involvement. As a result, Sustainability is becoming the new norm in corporate behavior, metrics and strategy as these companies develop supporting governance structure, system, policies and procedures. While the Sustainability programs are developed and deployed, a parallel process to integrate into risk management has been initiated by those in the natural resources or labor-intensive sectors (e.g., energy, manufacturing and consumer goods, etc.). These companies started to identify and manage Sustainability-related risks such as water scarcity, employment relations and supplier risk. The ripple effects soon expanded to banking, investment and insurance – industries that finance and insure their business activities to assess ESG risks.

This paper supports the effort to incorporate Sustainability into Enterprise Risk Management (ERM) with the focus of raising the awareness in the insurance sector. Sustainable ERM (SERM) adopts a capital-based approach to manage an insurer's overall risk profile within the capital infrastructure. In addition to the examination of financial capital, SERM examines the firm's utilization and its effect upon critical human capital and natural capital in order to manage stakeholder relationships with its employees, customers, the environment and the general public. As a result, SERM benefits from a broader purpose and outlook than traditional ERM. With continued evolution of regulation and legislation related to corporate governance and long-term Sustainability measurements, SERM will prepare for the company's business transformation while assisting in producing more effective and meaningful external disclosure including sustainability reports and integrated reports.

This paper provides perspectives as well as the preliminary framework and tools of SERM for insurers to facilitate the critical transition to integrate Sustainability into traditional ERM. Section 2 defines key terms such as Sustainability, SERM, and ESG. Section 3 identifies some of the key benefits of SERM in practice. A framework of how an insurer can measure and manage Sustainability, both qualitatively and quantitatively, associated with governance structure, is introduced in Section 4 with concluding remarks following in Section 5.

As we are in the early stage of recognizing the broad importance of SERM, this paper, as its name reflects, is an introduction to the concept. Much work remains to be completed to further develop the metrics and techniques for effective SERM in practice.

2 Key Terminologies

2.1 Defining Sustainability

Sustainability may be defined in several ways:

- Meeting the needs of the present without compromising the ability of future generations to meet their own needs (UN Brundtland report, 1987).
- The capacity to endure, or continue indefinitely.
- Preserving resources and energy over the long term.
- Providing sustenance and nourishment to keep in existence without diminishing.

For (re)insurers, Sustainability means *creating value consistent with the long-term preservation and enhancement of all forms of essential capital as part of the corporate objective.* By incorporating multiple capitals in the definition, sustainable business breaks away from the traditional mono-capital culture.

It is useful to recognize three broad categories of capital that are important to businesses: financial capital, human capital and natural capital. These correspond to three critical resources in the business processes: financial resources, human resources and natural resources. Sustainability can therefore be considered as *an objective that creates value consistent with the long-term preservation and enhancement of financial, human and natural capital.* Each capital can also have subcategories. Here capital should be considered as metaphors or means to broaden our perception on value creation.

Financial Capital

- Economic resources generated by financing, operating and investing to continuously support core business.
- Monetary assets to cover the economic effects of risk taking activities (in the insurance company this is economic capital).
- Monetary and physical assets as traditionally represented on a balance sheet (for some industries, physical assets such as factories, equipment and infrastructure may be singled out to form a subcategory of manufactured capital).

Human Capital

- Human resources, including people, institutions and relationships on which the health of the organization depends.
- Includes skills, knowledge, subject matter expertise, and knowledge-based tangibles such as models and analytical assets or other intellectual properties (may also be referred to as intellectual capital, a subcategory of human capital).
- Human relationships, employee engagement, trust and partnerships (this type of human capital is also referred to as social capital).
- Brand value (may also refer to as reputational capital).

Natural Capital

- Natural resources and processes needed by organizations to maintain operations, produce products and deliver services.
- Both renewable and non-renewable resources, e.g., plants, animals, air, water, soils, minerals.

Traditionally, insurers have invested heavily in the measurement and forecast of financial capital for the purpose of business management and financial reporting. However, financial capital does not exist in isolation as there are interdependencies with other forms of capital. For example, human capital is the foundation for an insurer's risk expertise as well as the driving force behind innovation and the evolution of markets. Insurers sell "promises" in the form of insurance policies that depend on the invisible currency of trust – social capital. Natural capital, including utility, water and office supplies such as paper-products, is also vital to an insurer's operations. Thus, corporate sustainability mirrors the conventional triple bottom line accounting framework - social responsibility (People),

environmental stewardship (Planet), and financial success (Profit). The relative prioritization of the triple bottom line or strategic deployment of capitals is driven by the corporation's mission, vision and values. Sustainability and underlying corporate purpose is the cornerstone of successful business as the organization's culture and ethical values are reflected in its use of and effects on the capitals.

2.2 Defining ESG

Often discussed in connection with Sustainability, the term ESG, an abbreviation for Environmental, Social and Governance, refers to a large set of extra-financial factors that affect the quality of a business. The investment community was the first to acknowledge the interconnectedness between financial risk and ESG risks. In the banking sector, the United Nations-supported Principles for Responsible Investment (PRI) was launched in April 2006 at the New York Stock Exchange to encourage companies to take a wider view of socially and environmentally responsible investing, thus generating long-term sustainable returns [2]. Increasingly, investors use ESG factors to evaluate corporate behavior and determine the future financial performance of companies. Table 1 presents examples of the broad type of factors that are considered under the umbrella of ESG. Many of these factors could relate to effectively managing financial, human and natural capital through strong governance practices.

Environmental	**Social**	**Governance**
• Climate change • Environmental compliance (on a legal level) • Environmental Health and Safety (EHS) for employees • Full accounting of externalities • Genuine interest in society • Reporting on environmental impacts and assuming responsibility for actions	• Employee relations • Employee rights • Community involvement • Customer loyalty • External stakeholder rights and involvement • Legal/regulatory breaches	• Anti-takeover provisions • Commitment to a wide range of external standards, principles & initiatives • Management performance relative to employees • Legal protection for investors • Strong shareholder/stakeholder protection commitment by company • Transparency

Table 1: Sample Factors Considered in ESG Analysis

Responding to the growing investors' needs, Bloomberg has been tracking more than 800 different metrics that cover various aspects of ESG from emission to shareholder rights. It offers terminal users the Bloomberg Intelligence analysis of ESG issues that can potentially affect the firms and sectors.

2.3 Sustainable ERM

Sustainability is becoming embedded in the corporate behavior, metrics and strategy of industry leaders driven by stakeholders' needs and regulatory requirements. Please refer to Appendix A for a detailed discussion of sustainability trends by these industry leaders.

With consumers and investors putting an increasing focus on Sustainability, traditional ERM becomes less effective in capturing the corporation's true risk, true cost and true value due to lack of a framework to evaluate ESG performance. To become a sustainable insurer, it is important to integrate Sustainability into core business and supporting functions. Thus, the global sustainability trends necessitate new definitions and measurements to protect corporate value and manage risk holistically.

Sustainable ERM, or SERM, is defined as the management of financial, human and natural capital for the purpose of stakeholders' value creation to realize sustainable development of the firm and therefore contribute to that of society. SERM is a necessary outcome of continued evolution of corporate responsibility and purpose-driven business. The following are the critical aspects of this definition:

Capital Management

- Comprehensive capital management entails financial capital, human capital and natural capital.
- Capital availability, quality and affordability affect long term viability of an organization's business model and capability of long-term value creation.

Stakeholders

- While the primary stakeholders are shareholders, SERM extends consideration of other stakeholders to include silent stakeholders (the environment and future generations).
- Leadership ethics in SERM ensure that no stakeholder is disadvantaged by the actions of others.

Value Creation

- Expanding the definition of the value beyond economic value to incorporate well-being and stewardship.
- Contributing to more intelligent, sustainable and inclusive growth that captures the true value of human and natural capital.

The multiple capital approach is not entirely new, especially in the sustainable development arena led by the UN. In the realm of business, Forum for the Future suggests a five-capital model while International Integrated Reporting Council (IIRC) promotes six-capital framework. While the number of capital categories may differ, the purpose is to mainstream sustainable business practice of environmentally friendly and socially responsible decision-making.

3 Benefits of SERM

Although ERM has gained traction and industry acceptance over the past decades, SERM is a new concept which requires higher human consciousness in conducting business to be regenerative of multiple capitals. The theoretical and philosophical construct of SERM based on multiple capitals is important in this paper since it is the foundation of subsequent development of tools and

methodologies to achieve the intended goals of the firm's sustainable development. For a detailed comparison between SERM and ERM, please refer to Appendix B.

Appendix A shows the relevance of Sustainability to the insurance industry as supported by global trends; therefore, the integration of Sustainability in ERM is imperative for the insurer's long-term success. Because of the holistic focus of SERM, firms can benefit from 1) comprehensive capital management, 2) improved relationship with stakeholders, and 3) sustainable development.

First, in terms of capital management, the SERM framework encourages development of a methodology to understand and measure values created across all vital capitals. This measurement allows for an assessment of the long-term viability of the business model and strategy through inclusive dialogues and KPI monitoring, and therefore informs decision-making in product enhancement, people strategy, and external communication. A survey as part of the Insurance Working Group of the United Nations Environment Programme Finance Initiative (UNEP FI) found that proper management of ESG factors could enhance insurance company earnings and long-term company value [3].

Additionally, the examination of all capitals enables the firm to realize the true value of its financial capital to support long-term value creation. For example, a company may consciously deplete financial capital in the short-run to enhance human capital through better workplace programs as well as automation and other improvements in its IT system. Studies show that initiatives to reduce the environmental footprint such as sound recycling practices and green building management have produced instances of improved productivity (human capital) and reduced operational cost, which allows for additional financial investments in natural and human capitals. According to these studies, the multiple-capital approach better shapes staffing and funding decisions, which optimizes resource allocation and methodology development.

Another benefit that SERM provides is an improved relationship with stakeholders. SERM allows for an effective means to manage the stakeholder relationship and intangible assets (including human capital and natural capital) through publishing of ESG factors and measurements. The firm's transparency, strategy and durability to attract multiple capital resources improves from the firm's introspective examination of its activities and stakeholders, and supports better decision-making for long-term value creation. The firm has the opportunity to build trust with its stakeholders through transparency and the future-fit value proposition as well as providing a buffer of credibility and sound reputation against potentially damaging events.

Finally, a key benefit of SERM is associated with sustainable development. We are moving into a world where solely generating profits is no longer sufficient to justify a firm's survival. The business model continues building social resilience and functioning as a force for good. Such a firm is seen as one of high purpose. The goals of business and goals of human well-being coalesce to deliver resilience, adaptability and creativity for our common future.

Evidence has shown that traditional ERM falls short in several crucial areas. Engineered to work backwards from traditional (short-term) financial performance metrics, its lack of emphasis on critical ESG margins underestimates the true financial impact of ESG performance. It is less effective in managing the stakeholder relationship and the firm's intangible assets, since these are often not included in the risk measurements. We see an underutilization of ESG data and information for commercial purposes, and lack of consistent and robust frameworks to combine information from various sources (financial vs. non-financial/extra-financial, hard data vs. soft data, tangible asset vs. intangible asset).

In sum, there are key reasons for an insurer to embrace SERM as an extension of its traditional ERM framework. For the Board and executives of the firm, it allows for the formulation of corporate strategies that simultaneously create economic, environmental and social value. It provides the Chief Risk Officer a holistic framework to manage enterprise risks, especially those that are traditionally considered to be 'un-quantifiable' in nature, while managers benefit from increased workforce productivity and satisfaction, having the ability to foster, nourish and protect human and intellectual capital. With the creation of SERM measurement metrics, risk professionals will better understand risk using new data (ESG data/big data) and enhancing tools for underwriting, reserving, investing and risk management. Although it is out of the scope of this paper, ESG data can be used for asset and liability management. Active participation of like-minded actuaries in constructing SERM is important to achieve the noted benefits.

4 Sustainable ERM Framework

Section 2.3 defines Sustainable ERM as the management of financial, human and natural capital for the purpose of stakeholders' value creation to realize sustainable development of the firm and therefore contribute to that of society. The need for SERM is clear and sustainability literacy is being developed for all stakeholders including ESG investors, employees and customers. This drives a trend to quantify non-financial performance or non-financial capital for disclosure and internal management purposes. It needs to be emphasized here that we did not just create new risk taxonomy of 'sustainability risk' under the traditional ERM framework, because the philosophy and guiding principles of SERM are fundamentally different from ERM.

A basic SERM framework focuses on quantifying and managing capitals of the organization; it addresses the potential overlap of Sustainability/ESG risks with other established risks, and manages non-financial capitals that are vital to financial capital.

This section outlines the building blocks of the preliminary SERM framework. It is important to note that the goal of SERM quantification is not to measure various capitals in monetary forms, nor does the framework provide a full account of complex interaction between the capitals to measure company's Sustainability. Quantification is a means to make sound business decisions. Equally important are qualitative analyses, expert judgment and vision of the company for the future and society at large.

4.1 Methodologies for Capturing Sustainability Information

4.1.1 Qualitative Approach

Companies may use narratives and descriptions to disclose the company's Sustainability practice in sustainability reports or integrated reports. Narratives are essentially stories to inform audiences on the role the company plays and how it creates value in addition to how much value it creates. The information is often subjective and anecdotal to capture the company's practice and value proposition on non-financial capitals. Examples include descriptions of waste management, sustainable procurement policies and discussion on ESG integration in investment and responsible business strategy with country-specific implementation plans. The qualitative approach is powerful to deliver the information in its totality compared to the reductionist approach of quantification.

4.1.2 KPI Approach

The Key Performance Indicator (KPI) approach is the most common method to manage and monitor a company's sustainability practices. Thanks to research done by institutions including The Natural Step and Future-Fit Business Benchmark, there is a good foundation of science-based sustainability principles and standards on which to base sustainability metrics. B Lab has a questionnaire that assesses through various indicators the sustainability performance of a prospective certified B Corp, which is a socially and environmentally responsible business. Because of the advanced regulatory framework on Sustainability in EU member states, there are many materials available in the European region. For example, the European Federation of Financial Analysts Societies (EFFAS) has developed ESG KPIs, including guidance for integration, for all financial sectors. Sustainability Accounting Standards Board (SASB) and the Global Reporting Initiative (GRI) have been refining existing reporting standards including ESG data disclosure. Table 2 provides examples of indicators used in insurers' Sustainability Scorecard for internal management as well as external reporting.

Natural Capital	• Premium volume of green insurance (for a list of products, please refer to [4]) • # of green solutions in asset management and insurance products • ESG investment ($, % total) on wind farm, clean tech, low carbon infrastructure • Physical unit of CO_2 emission (ton), water consumption (m^3), waste generated (ton), waste to landfill (ton) • Recycling rate, etc.
Human Capital	• Human capital performance such as return on investment (underlying earnings before tax + employee expenses)/employee expenses, value added (revenues – operating expenses)/headcount), productivity (employee expenses as a % of company revenues and financial impact (employee expenses/headcount) • % of employees who rate the company favorably on engagement index • % of employees who believe the company is a good corporate citizen • % voluntary employee turnover • % female employees & females on Board • Absentee rate • # of work-related injuries & illnesses • % of managed supply that has been engaged on the insurer's corporate responsibility • # of customer complaints per 1000 policies • Net Promotion Score (NPS), etc.

Table 2: Examples of Key Performance Indicators for Non-financial Capitals Used by Some Insurers

Note that it is possible to use appropriate ESG indicators as individual risk modifiers and underwriting risk modifiers if actuaries/underwriters think there is reasonable causal relationship to claims. For example, in professional liability, ESG factors such as employee turnover rate, quality of HR training, level of industry standards certification, documented risk management and loss prevention are used as rating variables. Traditionally underwriters have a set of ESG-related criteria to judge risk propensity to apply debits/credits. Governance factors such as quality of management and conflict of interest are common in Directors & Officers insurance ratemaking. In Surety underwriting, ESG represent the fourth "C" (Condition) to evaluate contractors for large infrastructures in addition to the traditional three "C"s – Capital, Capacity and Character. ESG risk assessment includes prescribed factors encompassing corruption, compliance, transparency, pollution and biodiversity. According to a survey conducted by the United Nations Environmental Programme Principles of Sustainable Insurance (UNEPFI PSI), underwriters also consider ESG factors such as forced resettlements and community health. [5] Allianz, Zurich, QBE and Swiss Re have implemented their own ESG underwriting guideline for selected industries. The UNEP is a process to develop global guidance to manage ESG risks in insurance underwriting with an initial focus on Property & Casualty business [6].

4.1.3 Monetized Quantification Approach

A monetized quantification approach is the use of scenarios to either simulate losses for areas where there are insufficient internal loss data or for simulating low-frequency, high-severity tail events. The following scenarios are for illustrative purposes only.

4.1.3.1 Scenarios

Suppose a global multi-line insurer is considering a scenario method to measure Sustainability. Examples of Sustainability scenarios can be constructed in the following forms:

1. Scenario 1: Failure to have efficient recycling practice

   Details 1: operational by-product is not repurposed efficiently to save money. Company's low-standard recycling programs alienate sustainability-conscious employees.

   Frequency (years) & severity (opportunity costs): 5 years - $1M, 15 years -$5M, 40 years- $35M.

2. Scenario 2: Failure to implement sufficient supply chain risk management

   Details 2: The ESG practice of suppliers is unchecked, leading to reputational damage or delayed delivery when unexpected negative ESG-related issues happen in the suppliers.

   Frequency (years) & severity: 5 years - $3M, 10 years -$10M, 35 years- $45M.

3. Scenario 3: Failure to pay employees and (sub)contractors fair living wage in local jurisdictions

   Details 3: Inadequate employee remuneration becomes a barrier to wellness and competence. Employees are not equally treated in compensation or opportunities, leading to loss of potential talent and increase in operational risk or even possible litigation for employment discrimination.

   Frequency (years) & severity: 5 years - $3M, 8 years -$4M, 15 years- $10M.

4. Scenario 4: Failure to develop and adhere to ESG underwriting criteria

   Details 4: The company offers surety bond to insure loss from non-performance and projects 5% annual growth in premium. Infrastructure projects could have associated ESG

risks such as environmental pollution, natural resource degradation, forced resettlement, poor working conditions and corruption, which are not systematically or wholly assessed by the insurer.

Frequency (years) & severity: 1 years - $2M, 5 years -$5M, 8 years- $9M.

Well-designed scenarios have the benefit of capturing diverse opinions, concerns, and experience/expertise of key professionals and incorporating Sustainability elements in a business model. Since scenarios (in return period loss) often depend upon subjective expert opinions, the challenge is that the abstract nature of the process can lead to unrealistic scenarios while lack of imagination can lead to underestimation. Actuaries involved in the scenario design need to understand the model limitation while striving to translate these opinions into a statistically acceptable construct. For example, the Exceedance Probability (EP) method can be used to simulate the annual scenario losses by fitting into the Poisson distribution and severity distribution. These Sustainability-related scenarios can be easily included in an existing economic capital model.

4.1.3.2 Internal Methodologies

Some companies use internal methodologies to quantify non-financial capital by combining quantitative and qualitative information to gain deeper insight. Some supplement traditional financial return on investment with environmental return on investment (eROI) and social return on investment (sROI) for holistic decision-making. Others may adopt a vendor's approach. There are many vendors and consulting firms offering customized solutions, metrics and reporting support. These include the Big Four accounting firms, management consulting firms such as Accenture and McKinsey, and sustainability-specialized firms such as SustainAnalytics, Natural Steps, TruCost and Route2Sustainability. For example, KPMG has developed the True Value Tool, which quantifies externalities. PwC's Total Impact Measurement and Management (TIMM) has a framework to monetize social, economic, environmental and tax impacts. The global efforts to shift onto the sustainable path are also evident in numerous open source resources to raise awareness and offer a platform for collaboration and tools to assist development of sustainable business. Table 3 provides an example of a human capital model piloted by Interface and Route2Sustainability [7].

Value of year-beginning human capital	Based on # of employees, their wages, their tenure years with the company, their years of formal education, and amount of internal training that the company has invested in them
+ annual investment in human capital	Based on fully-expensed new training and development; cost of employee volunteer time during the working hours; cost of medical and pension benefits; and cost of health and wellness benefits
+ annual appreciation of human capital	Based on value of step promotions; and level of employee engagement
- annual deprecation of human capital	Based on wages paid to employees over the year, cost of lost productivity as a result of sickness, absence, and health & safety incidents; cost of lower productivity during overtime worked; cost of lost productivity during turnover and cost of knowledge decay
= Value of year-end human capital	

Table 3: Example of a Human Capital Model Piloted by Interface Route2Sustainability

There are many public sources available to inspire the development of methodologies. Accounting for Sustainability (A4S) has issued guidance on natural capital and social capital quantification. World Business Council for Sustainable Development (WBCSD) has related protocol and toolkit. Both institutions have been working with leaders of various industries to tackle the measurement

challenges.

4.2 Governance of Sustainability/SERM

Good governance practice instills in the company the essential vision, process and structure to make decisions that ensure long-term sustainability. A sound governance structure, which consists of organizational structure, policies and procedures along with roles and responsibilities, is a necessary condition for a robust SERM program. It is an important requirement to have the support from the board of directors and senior management. It is from executive-level sponsorship that Sustainability initiatives will successfully be linked into the current governance structure, creating value for the company and benefits for all stakeholders.

Incorporating Sustainability into the company's fabric may be done over time in various stages. A basic approach that companies have employed is to create a Sustainability Committee to codify and quantify Sustainability risks across the organization. This is generally a stand-alone committee that starts the process of measuring Sustainability performance through KPIs developed in Section 4.1 and reports on the findings to the board of directors or other interested parties.

A more holistic approach has been put forth in a report by the UNEP FI Asset Management Working Group [8]. In this report a new governance model called "Integrated Governance" is introduced. Various phases to incorporate sustainability efforts within a company are described, with integrated governance presented as the end state or ultimate target of governance practices. The new governance paradigm requires full integration of Sustainability into the corporate strategy, with each traditional board committee integrating Sustainability issues into their charter. Decisions around Sustainability must be made at the top, with the corporate governance committee leading the charge. Table 4 illustrates how various committee roles can be augmented with Sustainability initiatives to create integrated governance. By incorporating this model of Integrated Governance, a company moves Sustainability issues from the periphery of corporate strategy to the heart of it.

Committee	Traditional Role	Additional Sustainability Role
Corporate Governance	Develop and monitor the company's governance principles.	Monitor and report on sustainability risks and opportunities.
Nominating	Oversee and evaluate the board's performance	Incorporate ESG targets within board evaluation.
Audit	Oversight of internal controls and audit of major functions; liaison with external auditors.	Ensure compliance with new sustainability regulations.
Compensation	Decide on the remuneration of executive directors/senior executives.	Link sustainability issues material to the business to ESG targets related to compensation.
Risk and Capital	Identify, assess and manage all categories of risk across a company.	Oversee enterprise ESG risk profile.

Table 4 Examples of Committee Roles Augmented with Sustainability Initiatives to Create Integrated Governance.

Under the integrated governance model, the SERM framework is built with the support from various committees based on the company's mission, vision and values. Corporate governance ensures better processes and infrastructure in place to enable multiple capital measurements and reporting. Companies can select KPIs developed by vendors, other institutions, or adopt internal methodologies as industry best practices emerge. Figure 1 is an illustration of a simple SERM framework. Early adopters would benefit from modernizing the company's IT and communication structures for Sustainability data, analysis and reporting in advance of many peers to prepare for the pro-Sustainability world.

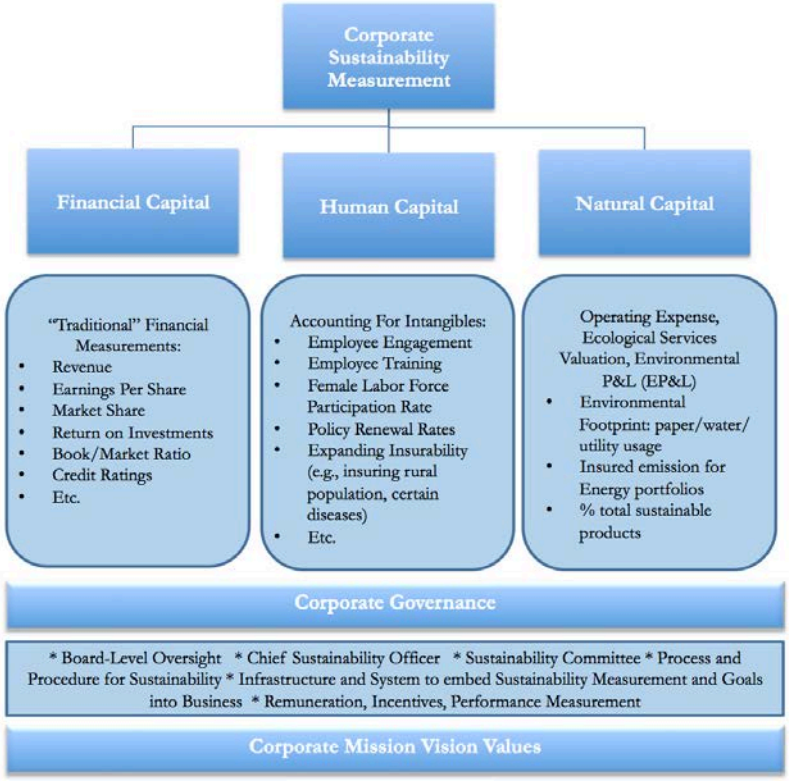


Figure 1 A Preliminary SERM Framework

As noted earlier, ESG risks from core business, i.e., underwriting, investment and claims, are not yet incorporated in the framework in this introductory paper. The next version of the framework may include the quantification of the company's impact as well as sustainability/ESG assessment along the value chain. Inspiration may be derived from the development of various sustainability scores used by rating agencies and third-party evaluators as well as on-going work at UNEP FI PSI.

5 CONCLUSION

The concept of Sustainability becomes increasingly important as society changes in response to the urgent need to move toward an environmentally, socially and economically sustainable future. SERM is a growing area and fits well into the concept of sustainable development by taking care of people and the environment. Done correctly, SERM will enable effective stewardship of multiple

capitals and capture ESG risk and opportunities. Disclosure of Sustainability measures from SERM offers additional insight into the quality of a company's management, culture, risk profile and other characteristics for stakeholders. Thus, the function of SERM is critical to corporate sustainability, which depends on the availability and quality of capital resources to the business. Going forward, it will become increasingly important for successful insurance leaders, especially actuaries, underwriters, brokers and other risk professionals, to develop Sustainability knowledge and ESG competency to inspire a global shift toward a sustainable future.

For future research, ESG integration in the core business of insurance may be closely studied to evaluate the insurer's environmental and social impact of its operation.

Appendix A Sustainability Trends

A.1 Stakeholders

Taking the perspectives of stakeholders in the insurance industry and beyond, we see multiple forces have led to the significance of Sustainability and its imperative for the future, where non-action or comfortable inaction will be no longer an option for a sustainable company.

Starting from within the enterprise, Millennials make up the growing cohort of current and prospective employees. A recent Gallup study shows that Millennials look for more than a paycheck; they want meaningful and gainful employment with organizations with purpose [9]. Their evaluation of a company also includes how the company impacts and improves the surrounding community in which it belongs. If a company's culture does not befit the beliefs of the future talent, it will have a difficult time attracting and retaining top talent. Examples of corporations taking action in this regard include sourcing materials from companies that have good sustainability practices and decommissioning products that contain materials that are harmful to the environment. Many committed firms have been requesting sustainability information from suppliers and business partners along the value chain. This includes filling out sustainability questionnaires and providing ESG scorecards for work bids. At the insurance company in which one of the working group members is employed, some large commercial clients are already asking such information.

To be fit for the future, companies have been adopting a sustainability strategy as a competitive advantage. Walmart is a good example. Perhaps a decade ago, Walmart was the most hated corporation in America, 'Saving Money' (for customers) at the expense of employees' fair wage and other exploitative strategies. The company was able to reposition itself out of the negative publicity to focus more on 'Living Better' for stakeholders by embracing Sustainability while engaging the business partners along its value chain. The company saved $3.4B from 2008 to 2013 by reducing packaging in its supply chain by 5% [10]. Now Walmart has been making progress toward its goal of being 100% powered by renewable energy, creating zero waste and selling products that sustain resources and the environment. The company has industry leadership in the Sustainable Appeal Coalition and Sustainability Consortium. This also influenced the value proposition of competitors like Costco, which has been refining its Sustainability practice and recently announced that it would intensify scrutiny of the products it carries for chemicals out of "regulatory and social concerns" [11]. Similar pro-sustainability corporate practice is driven by socially-aware health-conscious consumers who have also opened up the market for fair-trade products, non-GMOs and locally sourced food. Now the sustainability consumer is an important target market segment.

Another aspect concerns reputational risk. With the proliferation of social media and big data, information including negative ESG press travels faster and broader than ever before to various stakeholders. With the increase of Sustainability literacy in the general public, more and more people care about corporate's environmental and social impact in the process of making a profit. Managing the company's ESG risk is important in managing the reputational risk or protecting reputational capital. In this regard, insurers need to be more thoughtful in their internal and external communication. Active ESG risk management under SERM can enhance an insurer's crisis management or business continuity practice. In addition, offering insurance coverage to corporate clients without assessing whether or not clients violate international environmental and social standards may also expose the firm to serious reputational and compliance risks [12].

Another recent global trend is the increased emphasis on climate change and the implications on regional stability due to environmental and political issues. Often, enterprises are put into a position

to take a stance that could alienate a segment of their sustainability conscious clientele. Having a strong Sustainability practice supported by an SERM framework allows for a company to have a well-crafted commitment that can be communicated to and engaged with all stakeholders. With the recent developments of the Paris Climate Agreement, CEOs from many of the largest corporations in the world representing $17 trillion in assets have reiterated their continued commitment to climate change mitigation [13]. Other pressing Sustainability issues include environmental degradation, income disparity, plastic pollution and water shortage. Progressive firms have been aligning the corporate objectives and business practices to meet the Sustainable Development Goals (SDGs) to contribute to a more environmentally friendly socially equitable economy. In the long run, growing Sustainability-aware citizens and rising value-driven Millennials are likely to further the impact in policy-making and public domain (e.g., strengthening disclosure and governance standards).

While insurance companies do not heavily rely on natural resources or human labor, commercial clients in sectors such as manufacturing and energy have a large exposure to ESG risk. Insurers stand downstream of the consequences of unsustainable practices. For example, product liability and environmental liability loss are usually generated from covering products and operations that breach one or more ESG criteria. Policyholders' behaviors such as an unhealthy lifestyle and fatigue could trigger health and accident claims. Directors & Officers liabilities expose companies to risks associated with the decisions of insured corporations and executives with respect to sustainable business practices and disclosure of accurate information on these issues to stakeholders. The offering of the insurance products that encourage counter-sustainable behavior, when pro-Sustainability alternatives could easily be encouraged, exposes the firm to significant and unnecessary reputational risk. In this sense, insurers are directly affected by and indirectly responsible for their insureds' ESG damage and 'financed emission' by offering financial protection to these companies. ESG knowledge is, therefore, essential in understanding the quality of insured risks that influences insurer's financial performance. In the realm of Sustainability, financial performance is the lowest in the hierarchy because it is in fact the byproduct of non-financial performance.

Leading companies are positively influencing clients' behaviors to control potential ESG risks. For example, in one business transaction, Zurich discussed how it engaged with management of a construction client to ensure "responsible and sustainable business practice" [14].

As one of the key contributors to sustainable development, insurers can provide incentive for sustainable behavior by reducing the premium for conscious business and healthy lifestyles via Schedule Mod credits. They can extend their risk expertise to educate their clients to manage ESG risk profiles of the business. An example of such practice is for a company to "have effective responses by making decisions based on an ethical approach when it faces dilemma", where a "business transaction may be economically beneficial and perfectly fine from a legal and regulatory perspective, yet may have significant environmental or social downsides" [15].

Regulators and rating agencies also play an important role in shaping the insurance industry. The more developed a regulatory or legal framework for an ESG factor, the greater the influence the factor has on company operations.

The laws require various financial institutions to adhere to increased reporting of ESG performance including impactful activities on the communities around them, such as society, the environment, consumers and employees. In the US, financial accounting strengthening occurred with enactment of the 2002 Sarbanes-Oxley Act (SOX) after a series of accounting scandals including Enron, Tyco and WorldCom. Today, across the globe, organizations are under increasing pressure to meet more

sophisticated corporate transparency, responsibility and accountability standards for non-financial/ESG parameters. Notably, Europe has applicable statutory requirements and relevant codes of practice. Examples are the 'New Economic Regulations' Act (2001) in France and the Companies Act (2006) in the United Kingdom. These two laws impose requirements on companies to report on the environmental and social impacts of their business activities. Effective in 2017, companies with more than 500 employees in the European Union are required to disclose credible data and information on environmental, social and employee matters. The table below is a short summary of new laws related to Sustainability in various countries related to the financial services industry.

Area	Examples Relevant to Financial Sector
Banking	Brazil's Resolution No 4.327 (2014), Kenya's Sustainable finance Initiative (2014), China's Green Credit Guidelines (2012), Colombia's Green Protocol (2012), Nigeria's Sustainable Banking Principles (2012), Lebanon's reserves requirements for energy efficiency (2011) and Indonesia's Green Protocol (2009), etc.
Securities	Australia's Stock exchange reporting requirement (2014), EU's Directive on Disclosure of non-financial information (2013), France's Grenelle reporting Law (2012), USA's SEC climate disclosure guidance (2009), etc.
Investment	Malaysia's Investor Code (2014), Japan's Principles for Financial Action toward a Sustainable Society (2012), South Africa's Regulation 28 of the Pension Funds Act (2011) and UK's Pensions Act Reporting (1999), etc.
Insurance	UK's Prudential Regulatory Authority exploring climate change & insurance supervision (2014-5), USA's NAIC climate reporting (2009), etc.

Table 4 Sustainability-Related Policies in Different Countries

For other global initiatives covering ESG policies including metrics and disclosure, please refer to Black Rock's report [16].

Additionally, stock exchanges and bourses such as the Johannesburg Stock Exchange, the Australian Securities Exchange as well as bourses in Hong Kong, Singapore and Malaysia have included ESG/Sustainability disclosure as a listing requirement. This shows that the ability to assess a company's relative governance and performance in the context of non-financial factors is of great importance to institutional investors as well as private investors.

Sustainability Key Performance Indicators (KPIs) developed in SERM can be utilized for internal management as well as external disclosure to meet various stakeholders' information needs. Emerging standards led by the Sustainability Accounting Standards Board (SASB), the International Integrated Reporting Council (IIRC) and Global Reporting Initiative (GRI) are driving the needs and facilitating preparation for Sustainability disclosure. These new accounting and reporting principles not only support ERM to broaden the scope of value under consideration, but also help produce credible data for holistic decision-making under SERM.

In terms of ratings, Moody's has incorporated Sustainability in its credit rating since 2015 [17]. In the fall of 2015, S&P also launched the S&P Environmental & Socially Responsible Indices in response to clients' interest in socially responsible investments. In addition, there is a wide range of Sustainability rating agencies such as KLD, Sustainalytics, Trucos, GES, Vigeo, ASSET4 and Calvert. KLD ratings are among the earliest and most influential, especially in the US stock market, and are most widely used by researchers when compared with newer world-based ratings such as ASSET4 and GES. This has enabled positive development to facilitate global adaptation of these ratings

through the Global Initiative for Sustainability Ratings (GISR).

A.2 A Financial Perspective

One main justification for funding a Sustainability program is the enhancement of the financial bottom line: increase earnings and stock growth, lower insurance premiums, decrease borrowing costs and improve access to capital. Accenture, Deloitte, PwC, Goldman Sachs, *Harvard Business Review*, *MIT Sloan Management Review* and others have released data-driven case studies, global surveys and exhaustive reports that offer a compelling business case for Sustainability. In developing its own report, Morgan Stanley took into account a broad meta study conducted by Oxford University in 2014 [15] that reviewed academic studies conducted on the relationship between financial performance and Sustainability. Based on those results and others, the Morgan Stanley report made a strong case that "There is a positive relationship between corporate investment in sustainability and stock price and operational performance" [18]. It is discovered that financial markets value firms that practice Sustainability more, as "high sustainability firms significantly outperformed their counterparts" [19].

Based on the most comprehensive dataset on existing ESG–Corporate Financial Performance (CFP) research to date, Friede, Busch and Bassen (2015) aggregate evidence from 2000 empirical studies to establish the business case for Sustainability. Figure 1 shows a significant portion of the study shows positive relationship between the two.

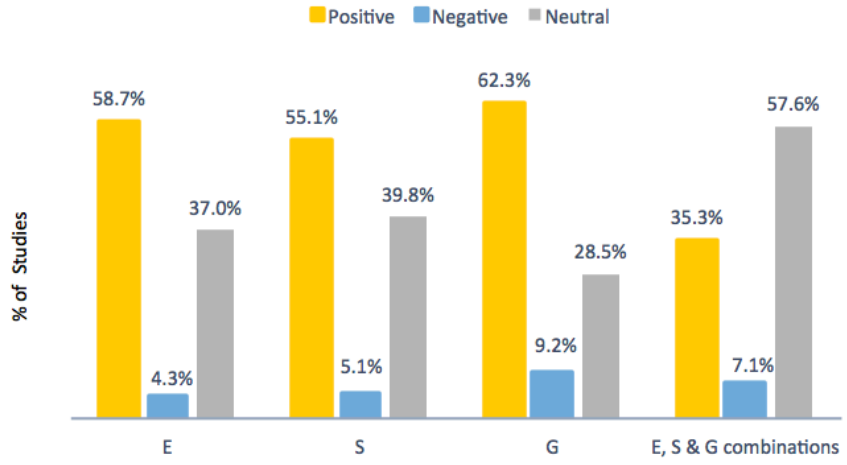
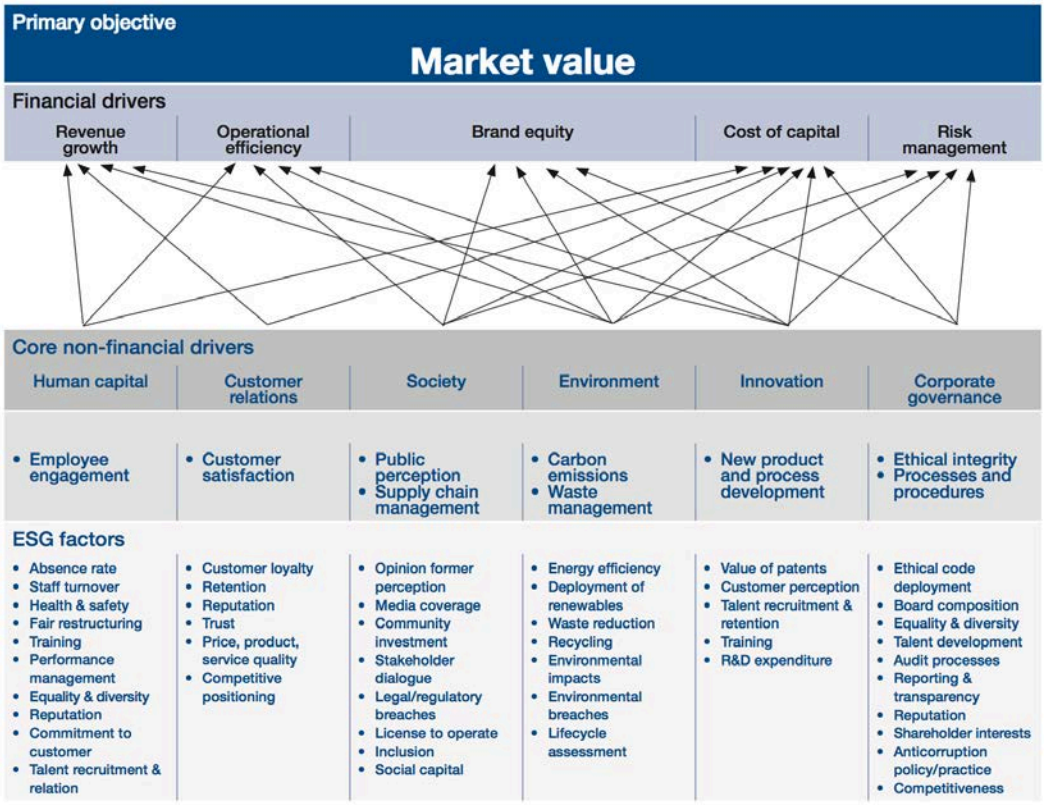


Figure 1 ESG Categories and Their Relationship to CFP

The European Academy of Business in Society (EABIS), in partnership with EU CSR Alliance Laboratory, studied the ESG-CFP linkage and mapped ESG factors to revenue-related outcome. The following diagram, Value Creation Framework, is created based on an extended literature review of more than 170 papers [20].



Those ESG factors arise from creation and/or usage of company's critical capitals. Thus the Sustainability assessment is a critical component of risk management to protect the corporate value.

Grant Thornton conducted a study to understand why companies fund Sustainability programs. The report reveals that the top driver towards more sustainable business practices globally is cost management, cited by 67% of respondents in 2014 up from 56% in 2011 [21]. Another study shows that people's willingness to buy, recommend, work for and invest in a company is driven 60% by their perceptions of the company and only 40% by their perceptions of the products [22]. This is likely an increasing trend: to fund Sustainability as a business enabler and strategic differentiator.

As a starting point, many Sustainability leaders have tackled attainable projects based on their unique ESG profile. Recognizing that not all initiatives are equal in terms of costs, efforts, and benefits, prioritizing and tackling these "quick win" projects show that an enterprise is thinking strategically as it embarks on a long-term journey of Sustainability. As an enterprise becomes more experienced and adept at implementing Sustainability related efforts, with feedback from various stakeholders and learning from prior efforts combined with technological advancement lowering the cost of resources over time, projects that were not financially viable before become more feasible and affordable. In addition, many positive externalities result from practicing Sustainability – intangible assets such as a stronger reputation and more positive brand recognition from being an enterprise that values

Sustainability and cares for its community, which often resonates well with local government and regulators, Millennials, and the future generation of workforce. Consequently, this would lead to improved employee engagement within the company and superior human resource cost efficiency.

A.3 Industry Leading Practice

Insurers provide a unique case in Sustainability since they not only manage their own risks from business activities but also manage the risks of customers while striving to remain profitable. The industry bears the financial consequences from internal inefficiency, unsustainable behavior and business practice of clients and partners all along the insurance value chain. Sustainability trends directly affect insurers' financial statements, which ties corporate success to the ESG performance of the company and their clients, as well as the activities of other agents not directly under the corporation's control. Hence managing the Sustainability-oriented activities of business units and clients, influencing behavioral trends in the marketplace, and monitoring those through an ESG assessment are vital to the long-term success of insurance companies [23].

Common themes across industry leaders can be linked to guidance provided by the United Nations Environment Programme Finance Initiative (UNEP FI). The UNEP FI exists to encourage systematic change in global finance to support a sustainable world, and guidance to insurance companies is through its Principles for Sustainable Insurance (PSI) which launched in June 2012 at the Rio+20 Summit. A company that adopts PSI can become a signatory of the Principles and a member of UNEP FI. The UNEP FI provides several action steps for each Principle that an insurance company can take in creating their SERM framework. As of year-end 2016, more than 100 organizations have adopted the Principles, including insurers representing approximately 20% of world premium volume and USD 14 trillion in assets under management.

The Principles, listed below with sample action items, are part of the insurance industry criteria of the Dow Jones Sustainability Indices and FTSE4Good. For a more complete list and detailed information, please refer to UNEP FI PSI website at www.unepfi.org/psi.

Principle 1: We will embed in our decision-making environmental, social and governance issues relevant to our insurance business.

Sample action items:

- Establish a company strategy at the Board and executive management levels to identify, assess, manage and monitor ESG issues in business operations
- Integrate ESG issues into recruitment, training and employee engagement programs
- Integrate ESG issues into the investment decision-making and ownership practices (e.g., by implementing the Principles for Responsible Investment)

Principle 2: We will work together with our clients and business partners to raise awareness of environmental, social and governance issues, manage risk and develop solutions.

Sample action items:

- Dialogue with clients and suppliers on the benefits of managing ESG issues and the company's expectations and requirements on ESG issues
- Provide clients and suppliers with information and tools that may help them manage ESG issues

Principle 3: We will work together with governments, regulators and other key stakeholders to promote widespread action across society on environmental, social and governance issues.

Sample action items:

- Dialogue with governments and regulators to develop integrated risk management approaches and risk transfer solutions
- Dialogue with media to promote public awareness of ESG issues and good risk management

Principle 4: We will demonstrate accountability and transparency in regularly disclosing publicly our progress in implementing the Principles.

Sample action items:

- Assess, measure and monitor the company's progress in managing ESG issues and proactively and regularly disclose this information publicly
- Participate in relevant disclosure or reporting frameworks

Common among these Principles is effective communication. As stated by the UNEP FI, "transparency is an integral form of accountability to the public, particularly in a voluntary and aspirational framework" [24]. Stakeholders within a company (e.g., employees) will benefit from understanding the goals of the above principles as it relates to their job responsibilities. Stakeholders outside of the company (e.g., investors and policyholders) will benefit from information shared as it relates to their own decision making. Adhering to the Principles will ensure a company fully embraces, and is a leader of, sustainable insurance practices.

Since ESG factors are relevant to both the insurance and investment operations of the insurance companies, industry leaders have also adopted Principles of Responsible Investment (PRI) for asset management. For more information on PRI, please refer to UNEP FI PRI website at www.unepfi.org/pri.

Appendix B

The guiding principles of SERM are compared against the standards of traditional ERM to evaluate this evolving means of holistic risk management.

While ERM focuses primarily (and often exclusively) on financial capital, SERM incorporates other forms of capital into risk management and decision making. As discussed in Section 2, in addition to the examination of financial capital, SERM examines the firm's utilization of, and effect upon, human capital (the firm's relationship to its employees, customers and suppliers), and natural capital (its relationship to the environment). As a result, the capital-based SERM approach benefits from a broader purpose than traditional ERM.

ERM's purpose is to assess, manage and monitor risks, optimize risk taking in relationship to financial strategic goals, keep risk level within the appetite, and satisfy the requirements of regulators and rating agencies. Extending this purpose, SERM requires that the firm understands the risks it faces through both financial and non-financial aspects of the business. It optimizes risk taking in relationship to strategic goals related to financial, human and natural capital. This includes the consultation with a wider range of stakeholders to incorporate their needs.

The strategic focus of SERM is consequently over a longer time horizon than traditional ERM. The focus of SERM is to develop a holistic picture of the entity's value creation story, particularly how the company generates value over the short, medium and long term in terms of the firm's investment in the multiple capitals.

The table bellows provides a comparison between Traditional ERM and SERM in terms of several broad criteria.

Criteria	Traditional ERM	SERM
Capital Considered	• Financial capital	• Financial capital, human capital, and natural capital
Purpose	• Assess, manage and monitor risks • Optimize risk taking in relationship to financial strategic goals • Keep risk level within appetite • Satisfy regulators and rating agencies	Assess, manage and nourish multiple capitals and optimize risk taking in relationship to Triple-Bottom-Line strategic goals • Identify and monitor KPIs across multiple capitals in order to optimize impacts, mitigate risks and improve performance
Strategic focus	• Identify and manage events and perils that may cause variation from the achievement of specific strategic goals • Strengthen financial capital • Help measure financial value and return on investment from the financial capital employed	• Understand interrelationship of various capitals to optimize business activities • Wider partnership within companies (e.g., marketing, HR) • Help develop full and holistic picture of the entity's value creation story - how the company generates value over the short, medium and long term in context of measuring return on the entity's investment in natural, human, and financial capitals
Leadership	• Tone from the top, require support from the Board to be successful	• Inspire growth, Net Positive in business operation, underwriting and investment • Nourish and cultivate human and natural capital as part of business activities
Risk Defined/Boundaries	• All risks the organization faces and generates with the focus on key risks • Exposure to any conceivable event or fact and resulting impact positively or negatively – variation from the expected	• All risks along the insurance value chain with special focus on ESG risk from clients, suppliers and other business partners which may impact ESG risk profile of the corporation
Accounting	• Financial accounting (fail to effectively capture intangible value)	• Develop green accounting, sustainability accounting for ESG issues. SASB standards are being developed

Criteria	Traditional ERM	SERM
Constraint	• Market short-termism detects organizational focus on financial capital • Organizational inertia	• Not universally accepted in the financial institutions largely due to lack of awareness • Lack of funding
Measurement	• Statistical/actuarial/economical models on data • Key Performance Indicators (KPIs) • Focused on more 'matured' risk (with more hard data and sophisticated modeling)	• Need to start with data collection: ESG matrices available from data providers, company's own decisions • Example of ESG indicator: financed/insured emission (% investment in fossil fuel, Energy client composite in underwriting portfolio) • Manage and measure environmental footprint • ESG data is available from data providers to measure non-financial capital
Data	• Mainly hard data (historical loss)	• Both hard data and soft data -include ESG data • Use big data for risk management
Reporting	• Financial reporting • Quantifiable	• Use integrated reporting to move beyond financial information alone to capture and communicate the full value of an organization • Quantified + Narrative
Value	• Economic value • Monetized	• Shared value • Produce by multiple capitals • Not necessarily monetized
Culture	• Risk-aware culture • Have 'risk owners' for accountability	• Culture to consider and manage environmental and social risks • Embed ESG performance matrices in remuneration • Encourage systems thinking in decision-making • Develop Sustainability literacy through employee training and internal advocate • Innovation and experimentation to build solutions

Criteria	Traditional ERM	SERM
IT	• Important to effectively manage risks – to ensure risk limits are followed • Business intelligence systems integrate enterprise data flows and generate analytic information for risk management decision making, internal controls testing and credit evaluation needs	• Build analytical infrastructure for information processing • Manage analytical asset including big data and relationship capital
Organizational Structure	• Risk management responsibility is decentralized and integrated into all levels of the organizations (aka Risk Management Function – RMF) • Chief Risk Officer	• While CROs should take additional responsibilities of Sustainability in terms of risk management, they should collaborate with CSO (Chief Sustainability Officer) if the companies have established a role • Broad oversight on Sustainability
Standards	• COSOII • ISO 31000:2009 • BS31100 (UK), AS/NZS 4360 (Australia/New Zealand) • Corporate governance (in some region like South Africa, social responsibility is one of the key characteristics)	• Standards are not compulsory or certifiable as of now • General Sustainability frameworks corporations can adopt include Sustainability Helix, Future-fit Business Benchmark, ThriveAbility framework
Asset	• Models • Use ERM to optimize business models and risk management	• Add to existing ERM intangible asset management
Communication	• Matrices and metrics are woven into reporting structures that engage the entire organization	• Collaboration along value chain • Stakeholders' legitimate needs and concerns are addressed in the integrated reporting • Progress is shown in selected KPIs

Criteria	Traditional ERM	SERM
Regulatory Requirement	• Basel II/III, Solvency II, Sarbanes-Oxley	• Various examples of policy innovation: EU Directive on disclosure of non-financial information, Japan's Principles for Financial Action towards a Sustainable Society, Australia's stock exchange reporting requirement, Brazil's Resolution No. 4.327
Rating Agency Expectation	• S&P, Moody's, Fitch, etc. evaluate risk management functions of insurance companies, take ERM into account when assigning credit ratings	• Still in development • Sustainability criteria/factors are being considered in evaluating credit-worthiness for certain industries.

REFERENCES

[1] United Nations, "Sustainable Development Goals," 2016, un.org/sustainabledevelopment/sustainable-development-goals/.

[2] United Nations Environmental Program Financial Initiative (UNEP FI), "Principles of Responsible Investment," 2006, unpri.org/pri/what-are-the-principles-for-responsible-investment.

[3] UNEP FI, "The Global State of Sustainable Insurance," 2009, unepfi.org/fileadmin/documents/global-state-of-sustainable-insurance_01.pdf.

[4] Zona, R., K. Roll and Z. Law, "Sustainable/Green Insurance Products," CAS *E-Forum* Winter 2014, casact.org/pubs/forum/14wforum/Zona_Roll_Law.pdf.

[5] UNEP FI, "2015 ESG Risk and Surety Bonds Survey," 2015, apfpasa.ch/eventos/2016-Asamblea-Cancun/pdf/programa/conferencias/miercoles/ing/4-b-Encuesta-Schiattone-&-Shea-2016-04-29-us-original.pdf.

[6] UNEP FI, "Global Guidance on the integration of environmental, social and governance risks into insurance underwriting," expected 2019. For details, please check unepfi.org/psi/underwriting-esg-risks/.

[7] The Natural Step of Canada, "Toward a Gold-standard Benchmark for a Truly Sustainable Business," 2014

[8] UNEP FI, "Integrated Governance," 2014, naturalstep.ca/sites/default/files/gold-standard-benchmark-for-sustainable-business.pdf.

[9] Gallup, "What Millennials Want to Work and Live," 2016. Access to the report is available at gallup.com.

[10] Sasine, Ron, "Walmart: Lessons Learned From A Commitment to Packaging Reduction," Packaging Digest, September 17, 2013, packagingdigest.com/sustainable-packaging/walmart-lessons-learned-commitment-packaging-reduction.

[11] Bloomberg News, "Costco Says It's Stepping Up Scrutiny of Chemicals in Products," June 15, 2017, bloomberg.com/news/articles/2017-06-15/costco-says-it-s-stepping-up-scrutiny-of-chemicals-in-products.

[12] Jaeggi, Oliver, "The Insurance Industry's Renewed Commitment to Sustainability," *MIT Sloane Management Review*, September 10, 2013, sloanreview.mit.edu/article/the-insurance-industrys-renewed-commitment-to-sustainability/.

[13] Winston, Andrew, "U.S. Business Leaders Want to Stay in the Paris Climate Accord," *Harvard Business Review*, May 31, 2017, hbr.org/2017/05/u-s-business-leaders-want-to-stay-in-the-paris-climate-accord.

[14] Zurich, "Mitigating ESG Risks in Underwriting and Investment," 2015. Report available at zurich.com.

[15] Clark, Gordon, Andreas Feiner, and Michael Viehs. "How Sustainability Can Drive Financial Outperformance." 2014. Final report released in March 2015 in conjunction with Oxford University and Arabesque Partners is available at arabesque.com/research/From_the_stockholder_to_the_stakeholder_web.pdf.

[16] BlackRock, "Exploring ESG: A Practitioner's Perspective," June 2016, www.blackrock.com/investing/literature/whitepaper/viewpoint-exploring-esg-a-practitioners-perspective-june-2016.pdf

[17]

[18] Moody's Investor Services, "Environmental, Social and Governance (ESG) Risks — Global: Moody's Approach to Assessing ESG Credit Analysis," October, 25, 2017, us.rbcgam.com/resources/docs/pdf/HTML-files/ESG/Roadshow/ESG%20in%20ratings.pdf.
[19] Morgan Stanley, "Sustainable Reality: Understanding the Performance of Sustainable Investment Strategies," March 2015, morganstanley.com/sustainableinvesting/pdf/sustainable-reality.pdf.
[20] Eccles, Robert G. and George Serafeim, "The Performance Frontier: Innovating for a Sustainable Strategy," *Harvard Business Review*, May 2013, hbr.org/2013/05/the-performance-frontier-innovating-for-a-sustainable-strategy.
[21] European Academy of Business in Society (EABIS), "Sustainable Value EABIS Research Project," 2009, bentley.edu/files/2015/04/15/Sustainable%20Value%20EABIS%20Research%20Project.pdf.
[22] Grant Thornton, "Corporate Social Responsibility: Beyond Financials," 2014, grantthornton.global/globalassets/1.-member-firms/global/insights/article-pdfs/2014/ibr2014_ibr_csr_web.pdf.
[23] Reputation Institution, "Global RepTrak™ 100 Survey," 2012, rankingthebrands.com/PDF/2012%20RepTrak%20100-Global_Report,%20Reputation%20Institute.pdf.
[24] Yang, Fan, "Sustainable ERM," *The Actuary*, Society of Actuaries, February/March 2015, 12:1, theactuarymagazine.org/wp-content/uploads/2016/12/act-2015-vol12-iss1.pdf.
[25] UNEP FI, "Principles for Sustainable Insurance," 2012, unepfi.org/psi/the-unep-fi-principles-for-sustainable-insurance/.

The CAS Working Group on Sustainable ERM (SERM) Members

Sandra J. Callanan, FCAS
George E. Davis, FCAS
Voon Seng Lai, FCAS
Tom Wakefield
Fan Yang, *Chairperson*

Property-Casualty Liability Estimation Reimagined

Christopher Gross, ACAS, MAAA

Abstract:

Loss development triangles are prone to problems such as mix shifts and changes in reserve adequacy. Much of the predictive information regarding the claims and exposures becomes lost through the act of aggregating data into triangles. In this paper, the author suggests a framework that includes objective establishment of actuarial case reserves, policy-level IBNR reserves, and other development components, mitigating such problems. While triangles still exist in this framework, the goal would be greater accuracy of claim and policy reserves so that adjustments to the total reserve would be minor. Greater information would be provided for reserve estimation, product pricing, and internal company management.

Keywords: loss development, triangles, case reserves, IBNR, predictive modeling, ratemaking, reserve allocation, individual claim development

1. INTRODUCTION

Triangle development techniques are used broadly for property-casualty reserve estimation and have been for a very long time. Their straightforward simplicity has made them indispensable, particularly when computing power was limited. But now, readily available computing power and techniques in predictive analytics make it possible to analyze claim development at a detailed level in ways that only could have been dreamed of in the past. Regardless of these advances, triangles perform an important step for the understanding and explanation of the loss development process, especially for non-actuaries, and are likely to remain part of the actuarial process for some time to come. By changing how the triangles themselves are constructed, this well-known actuarial framework can be made much more reliable, while shedding new light on the claims and policies being analyzed. In the process of rebuilding triangle elements from detailed data, the actuary can transform the industry toward a more detailed understanding of their income statements and balance sheets.

1.1 Research Context

The triangle development paradigm is well known and established, an excellent summary of which is provided by Friedland [1]. Advances in modeling detailed claim development in the context of actuarial reserving are discussed by Guszcza and Lommele [2], Antonio and Plat [3], and Korn [4].

1.2 Outline

The remaining sections of this paper are:

2. The Current General Process for Estimating and Booking Liabilities

2. THE CURRENT GENERAL PROCESS FOR ESTIMATING AND BOOKING LIABILITIES

2.1 Case Reserves

The general approach for estimating and booking liabilities is largely dependent on the establishment of case reserves at an individual claim level. Approaches for establishing case reserves vary widely from company to company but typically rely on a significant amount of subjective opinion on the part of claim adjusters, based on relevant information. This forms a significant portion of the overall liability for losses, and may or may be not accompanied by individual claim-level reserves for defense and cost containment or other claim related expenses.

The establishment of case reserves may be subject to guidelines maintained by the company. Sometimes more formulaic approaches may be taken. Often initial reserves (the reserves that are established when a claim is first opened) are given a common value until additional information and judgment can be applied.

2.2 Bulk/IBNR Reserves

In recognition of the potential for late reporting of claims after the statement date, as well as in recognition of the potential for reported claims to develop differently from what the case reserves indicate, companies book Bulk/IBNR reserves. Actuaries estimate the need for these reserves most commonly by analyzing aggregated historical data that reveals patterns of development for paid and case-incurred losses and loss adjustment expenses, both to cumulative amounts and relative to premium or other measures of exposure.

Recognition of differences in development across different types of exposures and claim types is managed by analyzing subgroups of claims/exposure that are considered to be relatively

homogeneous. This involves a tradeoff, as finer segmentation of reserve categories results in smaller volumes of data that may lack credibility or have patterns of development that are too variable to be reliable. More information on general approaches for estimating reserves using aggregated data are too numerous to mention here, but an excellent summary can be found in Friedland [1].

2.3 Unearned Premium Reserves

Property-casualty companies book an unearned premium liability for the portion of each policy's written premium that is associated with incurred losses that will occur after the statement date. Typically this liability is simply the portion of the entire policy period between the statement date and the expiration date of the policy, as applied to the total premium.

3. PROBLEMS WITH THE STANDARD APPROACH TO LOSS DEVELOPMENT

There are well-known problems with the estimation of liabilities under the current approach.

3.1 Shifts in Mix of Exposures

While attempts are made to include similar types of exposure in the definition of development triangle reserve categories, there are many dimensions that may develop differently. Some examples include geographical differences, class of business insured, size of account, deductible or limit, agency type, underwriting unit, etc. Because of the myriad dimensions, the actuary may not know what the different development characteristics may be across each dimension, and to know that a shifting mix of exposures may be creating distortions that render observed historical development patterns inappropriate for projecting future claim development. All potential differences cannot be dealt with effectively purely through reserve group segmentation alone, because the resulting triangles would be unreliable due to low volume.

Loss development triangles themselves are unlikely to indicate to the actuary that there has been an exposure shift until years have passed and the shift has resulted in new development patterns. By that time, significant damage may have been done to the balance sheet, and significant distortions may have been recorded in the income statement. Alternative methods (outside of the triangles themselves) are needed to indicate to the actuary that a) a shift has occurred in the nature of exposures, and b) that the shift is likely to change the development patterns.

3.2 Shifts in Mix of Claims

Similar to the shift in the mix of exposures, a shift in the mix of claims can also render the historical patterns inappropriate for projection. Whether it is due to changing environments for hazard or legal climate, or simply due to random occurrence, any time where case reserves carry with them differing levels of bias, a changing mix of claims will create distortions.

3.3 Changes in Case Reserve Adequacy

When case reserves are established subjectively, it is easy for changes in personnel or in the management of a claim department to have an impact on the adequacy of case reserves. This may be somewhat formal through new case reserving guidelines, or less formal through a changing claim handling culture. Environmental factors such as a changing tort climate can lead to a need for changes in case reserves, and case reserve adequacy can be eroded or supplemented if such changes are not immediately reflected in the case reserves. While actuaries look for evidence of changes in case reserve adequacy over time, they can be easily obscured by a changing mix of claims or exposures. It is likely unclear, when considering primarily aggregated data, whether a change in average case reserves is being driven by a change in adequacy or whether it is truly indicative of a change in expected future payments due to a different environment or mix of claims.

4. THE INCORPORATION OF PREDICTIVE MODELING TO ESTABLISH OBJECTIVE CASE RESERVES

There is growing attention currently in the actuarial profession around the area of individual claim models for establishing actuarial reserve estimates. Such models aim to look within the data that otherwise would be aggregated into triangles to build detailed models of how the individual claims develop over time[2] [3] [4]. This is a very important step for actuarial science, but is no small endeavor, and when such a model is complete, it is still necessary to reconcile the model with the triangle framework that is the standard in the industry. When differences arise with the traditional methods, confirmation bias can lead to the dismissal of these more detailed methods as being too new or too complicated. It would be ideal to incorporate the *information* gleaned from these new approaches *into* the very framework of the triangle standard approach for verification to resist such confirmation bias.

In triangle-based approaches to loss development, in order to avoid problems with changes in case reserve adequacy whether due to changes in mix of claims/exposures or due to a shift in case reserving philosophy within the claims department, the actuary could establish alternative case reserve estimates, based purely on objective characteristics. The ability to apply such a case reserving algorithm to

historical claims as well as current claims, provides the basis for considering loss development patterns that are more likely to be free from changes in adequacy. Berquist and Sherman [5] suggest a technique to apply current average case reserve levels to historical open claims. While this is certainly objective, the only claim characteristic that is considered in the approach is whether the claim is open or closed. If the current mix of claims is different from the historical mix of claims, the adjustment could be very inappropriate and lead to a worse estimate than without the adjustment. By using predictive modeling techniques we can develop more sophisticated alternative case-reserving algorithms.

Whether built from detailed claim life cycle models of the development, or built on existing information about claims, an actuarial case reserve estimate can be built that is consistent over time and based on objective claim and exposure characteristics.

4.1 Model Framework

In order to avoid problems with changes in case reserve adequacy whether due to changes in mix of claims/exposures or changes in treatment of claims by the claims department, the actuary could calculate alternative case reserves based on objective claim characteristics and the characteristics of the underlying exposure. Table 1 shows an illustrative example of the data organization used to build such a model.

Table 1. Illustrative Example of Objective Case Reserving Data

Claim ID	Age	Characteristic 1	Characteristic 2	Characteristic 3	…	Paid as of Age	Case Reserve as of Age	Paid as of now	Case Reserve Now	Adjustment to Case	Estimated Future Payments at Age
10001	1					700,000	1,060,000	1,910,000	-	1.07	1,210,000
10001	2					1,500,000	580,000	1,910,000	-	1.07	410,000
10001	3					1,900,000	120,000	1,910,000	-	1.07	10,000
10001	4					1,900,000	30,000	1,910,000	-	1.07	10,000
10001	5					1,905,000	10,000	1,910,000	-	1.07	5,000
10002	1					-	50,000	247,000	29,000	1.34	285,860
10002	2					220,000	35,000	247,000	29,000	1.34	65,860
10002	3					247,000	29,000	247,000	29,000	1.34	38,860
10003	2					500	60,000	37,000	12,000	1.34	52,580
10003	3					37,000	12,000	37,000	12,000	1.34	16,080

Three claims are shown in the table, at various points in their development during which the claims were open. The claim ID is shown, as is the age (i.e. month, quarter, or year) and the identifying characteristics of the claims. Also shown is the total amount of loss paid through each point of development, and the carried case reserve balance at that point. For each claim, the most recent valuation is also provided (paid losses to date and current case reserve). The 'Adjustment to Case' column reflects any estimated bias in the current case reserve. This adjustment will be discussed in greater detail in section 4.3. The final column in the table 'Estimated Future Payments at Age' is calculated as:

$$Estimated\ Future\ Payments\ at\ Age = Paid\ as\ of\ Now - Paid\ as\ of\ Age + Adjustment\ to\ Case \cdot Case\ Reserve\ Now$$

For claims that are already closed, this amount is sum of future payments as of the age, with perfect foresight. For claims that are still open, the most recent case reserve amount (adjusted) is added. This column becomes the target of the predictive model, and the variables to the left of the vertical line in the table (with the exception of the claim id) are potential predictive variables. Note that the case reserve itself as of the specific age of development is not included as a predictive variable. This would defeat the purpose of relying on objective characteristics[1]. Note also that the age of development is being used as a predictive characteristic. Other periods such as accident year or transaction date (i.e. calendar date or valuation date) should be used only carefully, because this could defeat the purpose of having a measure that is consistently determined over time. Values at various stages of development are *expected* to be different within the context of triangle-squaring and measured explicitly, so development age is appropriate as a variable. The assumption within triangle-squaring is that there are

[1] Other characteristics too may lack objectivity. There may have to be some consideration given to how much judgment is applied in setting the value of the characteristic. The more subjective it is, the more potential for problems with a lack of consistency over time.

not differences by accident period so this is generally left off as a potential variable, and is better handled by other variables reflecting a changing mix of claims/exposures.

Calendar period/transaction date should be used as a predictive variable only with caution. Trend/inflation *is* something that we would like to capture. Fluctuations in booked case reserves due to changing case reserving philosophy and practice *are not*. If including calendar period as a variable, care should be taken to constrain the behavior of the variable to be consistent with the capturing of trend. Alternatively the model could be parameterized with payments and reserves that have first been converted to constant dollar values, and then after the model-based actuarial case reserves are generated they can be adjusted back to appropriate nominal levels.

The paid losses to date as of the age of development are included here as a predictive variable because they are objectively known as of the point of prediction. Another variable that can be important is the amount of limit remaining for the claim.

A complete discussion of the topic of predictive modeling itself is outside the scope of this paper, other than to note that the usual best practices apply, such as using hold-out data to verify modeled relationships, valuing model simplicity, considering possible interaction effects, etc.

Interaction effects may be particularly important regarding those between development age and other variables as the speed of development may vary significantly across different types of claims.

Once a suitable predictive model is developed, it is then straightforward to apply it to every open claim at every stage of development in the history.

4.2 Data Elements

Any characteristic that is tied to a specific claim or its underlying exposure could be included in such a model. Some possible characteristics include geographical data, class code, cause of loss, injury type, age of claimant, nature of allegation, recent payment amounts, time since last payment, attorney involvement indicator, etc. Text mining techniques can be used to analyze free-form text fields such as adjuster comments.

Care should be taken to be aware of characteristics that may be changing over time. Information on such characteristics is usually provided to the actuary for the most recent evaluation only. Changing characteristics (such as the stage of the claim within the judicial or settlement process) can be very valuable in such an approach, if the evaluation of that characteristic is available at each age. Without having historical evaluations of that characteristic to place at the various ages, they should <u>not</u> be used, because applying the current evaluation to all past ages would be to say that this information would

have been known at all previous points in time, when it actually would not have been. This would create serious distortions in the analysis.

Assume, for example, that a characteristic value that only appears late in development is associated with serious cases with high payment potential. If that characteristic value is appropriately associated with claims only as it was actually identified, at the later ages, and does not occur earlier, everything will be fine. After the age at which the value occurs, the specific claims that have the value will have a higher actuarial case reserve estimate. Before the age at which the value occurs, that same potential for higher development will be shared among all the claims, because we do not know yet which ones will be so identified. Now contrast this with assigning that value back based on the current evaluation. At the earlier ages, claims would be inappropriately differentiated, with the claims that never received the value being identified as low-developing. Claims that are immature when the analysis is being performed would all be seen as low-developing, even though some of these eventually will receive the characteristic value.

4.3 Model Evolution

Little has been said about the "adjustment to case" factor to this point, other than that it reflects an estimate of the bias in the case reserves. This factor can vary dramatically in its usage from very simple to very complex.

At its simplest, this factor is 1.0 (no adjustment). This would say that we have no additional information available about carried case reserve bias. At its most complicated, it could reflect information gleaned from a separate individual claim model that considers claim characteristics and their impact on the life cycle behavior of the claim.

While using a factor of 1.0 is certainly the most simple, problems with mix shifts would still be an issue, because exposure types that are more prevalent in recent periods would also be at younger ages and any age differences could bleed into the exposure type variable. At the very least it would make sense to reflect the differences in case reserve bias that are understood through the existing triangle development framework.

Another approach would be to adjust case reserves based on analysis of report year paid and case-incurred triangles. By developing these triangles to ultimate, adjustment factors can be calculated as the ratio of (1-paid%)/(case-incurred% - paid%) at various evaluations. The report year triangles could be organized to reflect known differences in case development. This approach, while more accurate than no adjustment at all is likely to miss some of the observable differences between different types

of claims, because if triangles were subdivided among all of the important variables, they would soon lack stability and credibility.

It is worth noting that even with imperfections in the case reserve adjustments, there is still value in the approach, because closed claims form a part (often the majority) of the data, and because consistency over time is still obtained.

At the far end of the spectrum for the case adjustments is the building of a model of the development of individual claims over their life cycle as a function of claim and exposure characteristics. Detailed models can be built that not only estimate the total of future payments for a claim, but also the timing and variability of those payments. Models can be built based on first-dollar losses, with deductible impact, reinsurance, and other layer considerations identified explicitly.

The same approaches to reserve analysis that are traditionally used with loss development triangles would be performed on triangles using these revised objective case reserves (in combination with cumulative paid losses), the difference being that much less development should be observed in the aggregate, changes in adequacy are potentially avoided, and changes in mix are dealt with explicitly.

5. POLICY IBNR RESERVES

Instead of stopping with the idea of objective case reserves, we can develop an analogous concept to deal with late reporting of claims by developing policy IBNR reserves. These reserves would be equal to the expected loss amount for the policy based on its characteristics at the moment that a policy is written, and consistent with the expected loss reporting pattern, which also would be based on policy characteristics.

5.1 Model Framework

Compared to case reserves, establishing "true" IBNR[2] reserves at a policy level will often require many more data points to consider. Most policies will never have a claim, yet each would have a reserve at each point in time.

A complicating factor is that not only are we interested in the losses not yet reported for a given policy, but also we are interested in understanding the potential for unreported losses for different incurred dates within the policy. An estimate of the ultimate value unreported claims for this policy as

[2] For the remainder of this paper we will simply use "IBNR" to refer to estimates of claims that have been incurred but are not yet reported. As used here this does not include the provision for claims that have been reported but are expected to develop more.

of now is unlikely to be sufficient. We are likely to need to know how much of this has already been incurred (IBNR), and how much is associated with the unearned premium. Since accident periods are also of interest, we may need a further breakdown of the reserve by accident year, quarter, or month.

One approach to building such a reserve for each policy, with its appropriate incurred date detail, is to first organize reported claims for historical policies for a series of report lags. For example, the total of all claims (estimated ultimate value) for a given policy that were reported within 30 days of the incurred date. A number of lags would be tested. There is no reason for these lags to be evenly spaced, and good reason for them not to be, since the reporting activity is likely to decrease over time[3]. An appropriate series of report lags might be (in days) {1, 7, 15, 30, 60, 90, 180, 365, 365*2, 365*4,…}. Including every policy at every point in time would be quite large, and sampling is usually appropriate, to achieve a wide variety of policy characteristics and lags. Only policies that are fully earned should be included in this approach, but there is no requirement that all the losses would be reported at the point of measurement, but only lags that would be completely observable for a policy should be included (e.g. a policy with an expiration date of 70 days ago could be observed for report lag claims of 60 days, but not of 90 days.)

The predictive model seeks to predict the reported losses for a given policy and a given reporting lag as a function of the written premium, the reporting lag itself, and all of the available policy characteristics (i.e. rating variables). Note that claim characteristics are not included in this model because before the claim is reported its characteristics are not yet known. As the reporting lag increases, the model simply indicates losses as a function of premium and account characteristics (expected loss ratio). The impact of the reporting lag variable itself reflects the reporting pattern, and potential interaction effects between that variable and the policy characteristic variables should be investigated, reflecting potential difference in reporting patterns across different types of exposures.

Once such a model has been built, we can estimate the "true" IBNR for any policy/accident period combination. Let $P \cdot R(l, \boldsymbol{\theta})$ represent the expected reported (ultimate-value) claim cost amount for a given policy with premium P, report lag l, and, and policy characteristics $\boldsymbol{\theta}$. $P \cdot R(\infty, \boldsymbol{\theta})$ then represents the expected total claim cost associated with the premium. If we assume that loss potential is proportional to earned premium then we can substitute earned premium in this equation for the portion of the policy premium that is earned in any particular accident period. The IBNR estimate for a policy, for a given accident period at a particular point of evaluation can then be calculated by integrating the expected reported losses over the incurred dates included in the accident period and

[3] This may not be true for all lines. For example with exposures that have a deadline such as warranty or where statutes of limitation or repose are particularly important there may specific "end-point" lags to consider.

subtracting from the total expected loss (reflecting the range of lags associated with range of incurred dates).

With a reasonably granular choice of a report lag series in the predictive model, and with premium earning and loss incurral constant over the period of a policy, the estimate can be approximated by:

$$EP \cdot \left[R(\infty, \boldsymbol{\theta}) - 0.5[R(l_0, \boldsymbol{\theta}) + R(l_1, \boldsymbol{\theta})]\right]$$

where EP is the earned premium, l_0 is the lag associated with the age of the end of the accident period at the point of evaluation, and l_1 is the lag associated with the age of the beginning of the accident period at the point of evaluation ($l_0 < l_1$).

For an estimate of the losses associated with unearned premium we have:

$$UP \cdot R(\infty, \boldsymbol{\theta})$$

where UP is the unearned premium,

and the estimate of the losses for a policy newly written are:

$$WP \cdot R(\infty, \boldsymbol{\theta})$$

where WP is the written premium.

By building a model of loss development by incorporating claim and policy level detail, we automatically build a pricing model as well, but one that reflects the expected differences in claim development in the currently observed losses.

5.2 The Incurred Triangle

With the ability to provide an IBNR estimate for every accident period and valuation date by policy, and combining this together with actuarial case reserves and cumulative paid losses, we arrive at an estimated ultimate incurred loss at every point in the triangle, but one that reflects differences in mix and that attempts to remove any fluctuations in case reserve adequacy. It should exhibit little upward or downward development in the aggregate, but rather random fluctuation across the individual accident periods.

This incurred triangle is not to be confused with triangle referred to as an incurred triangle commonly in actuarial practice = cumulative paid loss + case reserve balance, more appropriately referred to as a case-incurred triangle.

This new incurred triangle serves as a check on the predictive models that have built the reserves that are embedded within it. If they were built well, the lack of development over time illustrates their

stability and gives support that the actuarial case reserves and policy IBNR reserves are not expected to exhibit future development. If the incurred triangle does show upward or downward development, that development can be dealt with using usual triangle development methods. At the very least the adjustment should be smaller, and the need for reserve allocation to understand the true results of subsets of exposure, smaller.

5.3 Model Evolution

As with the actuarial case reserve, an evolution of models for policy IBNR is likely. Layers, variability, and timing are all worth consideration as such models continue to improve. Models that explicitly estimate differences in reporting lag, frequency and severity add additional strength to such a model.

6. OTHER DETAILED ESTIMATES

Other types of detailed estimates can be valuable within this framework. Two that we will discuss here are claim reopen reserves and salvage/subrogation assets. These would be estimated prior to the building of the policy IBNR model, and would form a portion of the inputs into that model.

6.1 Claim Reopen Reserves

This reserve would be at the claim level, for each claim currently closed that is considered to have a potential to reopen. It will likely be practical to set a cutoff for claims that have been closed more than a certain number of periods. The data to parameterize this model would consist of each previously closed claim at each age of development, with the variable to be predicted being the total payments after that date plus the current objective case reserve (if the claim is currently open).

6.2 Salvage/Subrogation Asset

This asset or contra-liability would be at the claim level for all claims that have paid losses to date and are considered to have a potential to see future salvage/subrogation. For this category as well, given the large set of closed claims, it will likely be practical to set a cutoff based on how long the claim has been closed. The data to parameterize this model would consist of each claim at each age of development, for only those claims that have paid losses as of that age of development, with the variable to be predicted being the recoveries, and the amount of payments to date providing the exposure to these future payments.

7. BENEFITS TO OTHER AREAS OF ACTUARIAL/OPERATIONAL PRACTICE

The advantages of actuaries being directly involved in understanding claim development in all of its facets (reporting lag, adjustment, interim payments, final payments, recoveries, etc.) and understanding the differences in these facets across the various claim and exposure features carry well beyond the establishment of an aggregate reserve level.

One of the most significant areas that benefits is pricing. Far too often broad assumptions are made about loss development with pricing models, whether they use sophisticated predictive modeling or more traditional ratemaking techniques. Often it is implicitly assumed that all undeveloped claims will develop similarly. This is rarely the case, and by including exposure/rating variables into the process of generating actuarial case reserves and policy IBNR, the actuary can explicitly test this assumption and where incorrect, adjust accordingly.

A closely related area that would benefit is in internal management reporting. A typical part of the process at insurance companies for building internal management reports is to allocate booked bulk/IBNR reserves to a more detailed level (product/office/agency, etc.). Often actuaries are involved in setting the process for this allocation, and it is treated as an afterthought to the reserving process. Typically anomalies can occur in this process and inappropriate allocations can have real and destructive impact through the impact that they have on salary and bonus incentives for management. By building such estimates at the detailed level, there is much less that is left to broad allocation. The signal of profitability as estimated by the actuary will translate much more quickly into operational decision-making. This new framework also makes it very easy to transition from building internal management reports around earned premium and estimated associated loss to building those reports around written premium, by explicitly considering the estimated losses associated with the unearned premium reserves.

To the extent that there are differences between the actuarial case reserves and actual case reserves set by the claims department, the information may be valuable to the claim department itself, potentially resulting in the improvement of claim handling based on the predictive modeling performed by the actuary.

8. SUMMARY DISCUSSION

Much can be gained by building actuarial reserve estimates at a much more granular level than historically has been done, and then using triangles to handle only the residual development shown

on actuarial case reserves and policy level IBNR. While this represents a sea change in actuarial reserving analysis, requiring considerably more analysis than the historical reserving paradigm, the benefits are significant, not only to more quickly identifying issues with changes in mix of claims and exposures and correcting for changes in case reserve adequacy, but also dramatically improving actuarial pricing models and insurance company operations.

REFERENCES

[1] Friedland, J.F., "Estimating Unpaid Claims Using Basic Techniques, Casualty Actuarial Society", Third Version, July 2010
[2] Guszcza, J and Lommele, J, "Loss Reserving Using Claim-Level Data", *Casualty Actuarial Society Forum*, Fall 2006
[3] Antonio, K and Plat, R, "Micro-Level Stochastic Loss Reserving for General Insurance", October 2012
[4] Korn, U, "A Comprehensive, Non-Aggregated, Stochastic Approach to Loss Development", *Variance* 10:1, 2016, pp 13-33
[5] Berquist, J and Sherman, R, "Loss Reserve Adequacy Testing: A Comprehensive, Systematic Approach", *Proceedings of the Casualty Actuarial Society* (PCAS) LXIV, 1977, pp. 123-184.

Biography of the Author

Chris Gross is the president of Gross Consulting. He has 25 years of experience in the insurance industry. He has served a wide variety of clients across the United States, from Fortune 500 companies to startups, including insurance companies, banks, manufacturers and governmental entities. He has provided clients with cutting-edge analytical services as well as unique software solutions. Chris has pioneered the generating of actuarial reserve estimates using predictive models of claim-level development. Before starting Gross Consulting in 2005, Chris was VP & CFO of the Financial & Professional Services business group at Travelers. Chris has bachelor's degrees in mathematics and economics from the University of Minnesota.

Actuarial Models in the Language J

Richard L. Vaughan, FCAS

Abstract. This paper accompanies and explains a library of actuarial models, written in the language J, which has been placed in the public domain and uploaded to the CAS web site. Readers are encouraged to use and enhance these models and to add new ones. The paper also provides an introduction to J, which the author considers remarkably well adapted to expressing and solving actuarial problems.

1. INTRODUCTION

1.1 Actuaries and Programming Languages

Many actuarial operations involve algorithms of moderate complexity applied to data structures of moderate complexity, but where neither the algorithms nor the data structures are widely used elsewhere. Actuaries are on their own in finding ways of performing these operations efficiently.

Fifty years ago much of the work in an actuarial office was done by filling out successive columns of paper spreadsheets. Data for larger projects was kept on punch cards, much like those used in the 1890 census, which were sorted and tabulated in overnight jobs by mechanical devices in a central office. This environment began changing in the 1970's, as desktop computers appeared, first, for business users, from Olivetti and IBM, and later, for both business and the general public, from Apple, IBM, and many others. Standardization to a handful of operating systems created a large market for commercial software. This software naturally catered to the broad needs of the business community rather than the niche requirements of actuaries. However, there were, among the emerging software products, certain tools that actuaries could use to create their own solutions. These included a wide variety of programming languages and several "electronic spreadsheets".

Individual actuaries responded to these developments by turning to languages such as C, Pascal, BASIC, and APL, or to spreadsheets such as Visicalc, Lotus 123, or Excel. Consulting houses responded by producing large-scale programs for repetitive tasks such as loss reserving, first for their own use and then for the market. Insurers responded by licensing such programs for their actuarial departments and providing personal computers and software for their employees.

In recent years Excel has become the *de facto* standard spreadsheet. It is installed on nearly every computer in nearly every office, so that one may assume that its files will be readable by nearly any correspondent. Many actuaries have responded to *this* development by forcing all their work into an Excel framework. As a result, fewer actuaries use general-purpose languages to define and solve their problems and to create collections of tools for future use. This is unfortunate, for Excel is by

no means the most satisfactory platform for actuarial work. It does many things well, but it is still a spreadsheet, and imposes a grid format, with every step visible, on problems that might better be expressed in languages that are both more flexible and more discriminating as to which details they expose and which details they execute out of sight. Excel's worksheets are not easily scalable, they make it difficult to follow program logic, they impose size limitations unrelated to memory resources, and they do not clarify the work by distinguishing among cells used for data, parameters, intermediate results, and output.

Insurers continue to use commercial actuarial programs for routine work. In the interest of uniformity they accept whatever limitations any particular such program may have. The programs themselves are very capable and address issues difficult for individual actuaries to invent anew, such as estimating the covariance among lines in loss reserving. Their polished user interfaces and predefined output formats are excellent for the jobs for which they were written, but may limit their ability to adapt to new, perhaps only slightly different, problems.

We think that, if more actuaries returned to the habit of coding their own algorithms, in languages more flexible than a spreadsheet, it would improve the quality of their work and the professionalism of the actuarial community.

1.2 Thoughts on R

One language which *has* attracted the attention of actuaries and actuarial departments in recent years is R. In fact, R has become the most prominent language for work by or for the CAS and its Working Parties. So it behooves us to say a few words about R before turning to J.

Any general-purpose language can express any problem that any other such language can express, and R is no exception, so we might ask why the choice of language makes any difference. The answer lies in issues of style: conciseness, readability, scope of names, convenience of syntax, ease of parameter passing. It also lies in the development environment: editors, interpreters, compilers, debuggers. And it lies in the historical acceptance of the language, and the availability of pre-programmed and pre-tested solutions to parts of a problem that may be plugged in to contribute to a solution of the entire problem.

In many of these ways R excels, and it especially excels Excel. Its syntax is convenient for function definition and parameter passing. It is an interpreted language, which allows the user to work interactively in real time. And it has an enormous collection of packages, or code libraries, contributed by users. Many of these are statistical, and a few are specifically actuarial.

But R is not the only language with useful properties for actuaries. Nor is it necessarily the best, especially if the actuary expects a language to assist him or her in visualizing operations on arrays of

data and expressing them concisely and evocatively. There exists a family of languages, the APL family, that is practically unexcelled in these respects; it might well be called the "Actuarial Programming Language" family. The present-day scion of this family is J.

1.3 Thoughts on J

We shall go into J in greater detail presently, but first a few general observations are in order. For actuaries, perhaps the most salient characteristic of J is that it operates on arrays as its basic objects. These arrays, which J calls *nouns*, may be *atoms* or scalars (think dimensionless numbers), *lists* or vectors (think development factors), *tables* or matrices (think loss triangles), 3-dimensional arrays (think simulated loss triangles stacked one above another), 4-dimensional arrays, and so forth.

The number of dimensions of an array is its *rank*. Functions, called *verbs* by J, operate on arrays, and *functions also have rank*. The rank of a function is the rank of the array the function naturally "works on". For example, if we have a function `f` that returns a vector of development factors from a loss triangle, then `f` has intrinsic rank `2`, or, more precisely, will normally have been assigned rank `2` when it was defined. If applied to a rank-`3` array `A` of 10000 simulated loss triangles, `f` will operate on each triangle separately, treating `A` as a vector of 10000 *cells* each of rank `2`, and will line up all the results in a *frame* of length 10000, producing a matrix of 10000 vectors of development factors. The user does not need to code any loops or use any indices to achieve this extensibility. The loops are *there*, but they are handled invisibly, seamlessly, and efficiently by the interpreter.

Verbs in J may have either a right argument alone or both right and left arguments, and the same name may be used for one function of each kind. Think of `-`, negation, and `-`, subtraction.

In addition to verbs, J has operators called *adverbs* and *conjunctions*., which take functions or nouns as their arguments and return functions as their results. Adverbs and conjunctions let us modify functions, and conjunctions let us compose them, in a wide variety of ways. Moreover, J has special constructs known as *hooks* and *forks*, which compose functions in yet more ways. In this manner J lets us generalize the notion of the composition of two functions to all the cases implied by the fact that functions may have either one or two arguments, and many more cases.

This flexibility of function composition facilitates what J calls *tacit* definition, or functional programming, in which new functions are defined entirely in terms of existing functions with no mention of their arguments and no use of local variables. This allows many functions to be defined with extraordinary compactness.

Functions in J are exceptionally scalable in the dimensions of their arguments and results. A loss development program originally written for a 10 x 10 annual triangle may, without any modification, apply to the following year's 11 x 11 triangle or to a 40 x 40 quarterly triangle. Such scalability is

possible with careful coding in languages such as R, but it is difficult to achieve in spreadsheets. In J it is almost automatic.

Users may define new objects of any type. The naming conventions of J make it impossible to confuse user-defined objects with system primitives, and they give the user complete freedom to reassign names on the fly, even to objects of different kinds and to nouns of different types, ranks, or shapes. No declarations are necessary.

Overall, J is uncompromisingly consistent and elegant in its treatment of functions applied to arrays. This makes it a very solid foundation for actuarial applications. More importantly, J guides the user to a holistic understanding of problems and solutions, unencumbered by administrative details such as indices in loops and limits of summations. The actuary who becomes fluent in J will have a "secret weapon" allowing him or her to complete tasks in record time. He or she may then treat each problem as an instance of a class of similar problems, and spend some of the saved time generalizing the programs developed for one instance to permanent and tested tools suitable for the whole class, accumulating a library of programs for the efficient solution of future problems.

1.4 Library of Models

Accompanying this paper, and accessible on the CAS web site, is the J *script* **act.ijs** containing the source code of the *beginning* of such a library of actuarial models. A script is a simple text file of J sentences, readable and editable by any text editor. The editor supplied with the J system is especially well suited for this purpose, as it uses color to distinguish objects of different types. When loaded by the J interpreter, **act.ijs** defines a collection of objects, ranging from top-level functions producing large reports to small components or utility functions useful in constructing both the higher-level objects in this library and new objects defined by each user.

The reader is encouraged to visit **Jsoftware.com**, download and install the free interpreter, run it, download the available packages, some of which we use (menu **Tools|Package Manager**) and explore the supplied documentation. To do this, click on **Help|Vocabulary** to reveal a table listing all the primitive objects of J and giving a precise definition of each. Next click on any one of the items in the new top menu bar; it is probably best to start with **Pri** (Primer), **LJ** (Learning J), **JfC** (J for C Programmers, by Henry Rich, an excellent tutorial for anyone), and then **Dic** (Dictionary, a comprehensive description of J). Once acquainted with these resources, but leaving close perusal of them for later, the reader should begin experimenting with J interactively.

Finally (that is, after a couple of hours), load and run the script **act.ijs**, and experiment with some of its functions. Each function, and much of the internal logic of the larger functions, is documented with comments, which in J are preceded with the word `NB.` (*nota bene*, note well). In

the case of tacitly definitions, which do not mention their arguments `x` and `y`, our comments still refer to the right argument as `y`, to the left argument, if any, as `x`, and to the result as `z`.

Eventually we hope you will define new functions and send some of them to the CAS in supplementary scripts to be made available to the wider community. If you have suggestions for changes to the original code in **act.ijs**, send them to the author via email and he will evaluate them and modify the main script as appropriate.

2. SYNOPSIS OF THE J LANGUAGE

2.1 APL and J

In the mid 1950's, a young assistant professor at Harvard, Dr. Kenneth Iverson, from rural Alberta, began work on linearizing the notation of applied mathematics, so that it might be more easily translated into machine language and more intuitively understood by a human reader. He wrote a little book, *A Programming Language*, describing his concepts. The story goes that when Iverson came up for tenure, the senior professors noted that all he had published was "one little book". But in 1979 that little book won for Iverson computer science's highest honor, the A.M. Turing Award, "For his pioneering effort in programming languages and mathematical notation resulting in what the computing field now knows as APL, for his contributions to the implementation of interactive systems, to educational uses of APL, and to programming language theory and practice".

Iverson moved from Harvard to IBM, and, by 1963, he and his colleagues had implemented APL on an IBM 1620 computer. In subsequent years, IBM and several other companies marketed APL interpreters and time-sharing systems, which extended APL in various directions. Iverson himself worked for one of these companies, I.P. Sharp Associates in Toronto, from 1980 to 1987. The extensions of APL created by Iverson and his associates at Sharp paved the way for the comprehensive treatment of functions applied to arrays that characterizes J. By 1990, Iverson, Roger Hui, Arthur Whitney, and others created the first version of J, incorporating the latest theoretical extensions of Sharp APL together with rank and forks. Iverson passed away in 2004, but the development of J has continued under the leadership of Roger Hui. J is available at no cost for Windows, MacOS X, Linux, Android, and Raspberry Pi, under a GNU public license. J has an extensive user community contributing suggestions and improvements through forums.

While J is a member of the APL family, and a direct descendant of the original 1963 APL, it is different in several respects, including the default order of axes to which certain functions apply and the management of local and global variables; APL and J code cannot easily be translated one to the other. Moreover, APL uses a unique character set, which once earned it a special "ball" on the IBM

Selectric typewriter. These characters allow many functions to be represented by a single symbol, and they have mathematical and/or heuristic connotations which have made them much beloved of APL users. But they proved difficult to set in type, or to communicate electronically using the 7-bit ASCII character set once common among online messaging services. To avoid these problems Iverson restricted J to the low-order ASCII characters. But he did not give up on special symbols! By spelling the names of primitive objects either entirely or partly with ASCII *punctuation marks*, Iverson eliminated all possibility of confusion with user-defined names. By allowing many symbols to be used as J words either alone or when followed by `.` or `:` (which he called *inflections*), he created an even larger number of words for primitive objects than those of APL, albeit many of them digraphs (or occasionally longer) rather than single symbols. The use of digraphs opened the possibility of having several closely related operations represented by similar-appearing words. Despite this need for two characters to represent some primitive functions for which APL needed just one, the J language is even more concise than APL, because J's treatment of function rank avoids lengthy circumlocutions, and because J supports tacit definitions.

It has been said that J is what Iverson would have made of APL if he had had it to do over again. In fact, he and his colleagues did do it over again. They took something already very elegant and powerful and rationalized and extended it with the benefit of thirty years' research and hindsight. By putting J in the public domain they have given the computing world – including students, educators, and not least actuaries – an example of precise and coherent thought seldom found in commercial products of our day and age.

2.2 J Vocabulary and Syntax

The easiest way to learn the syntax of J is to begin using it, interactively, to solve the simple problems that arise daily in an actuarial office and that the reader might otherwise be tempted to do in Excel. In this way one will simultaneously learn the vocabulary of J, its syntax, and its provisions for defining new objects, a bit (or perhaps a few dozen bytes) at a time. We hope our introduction here will help the reader get started.

J is an interpreted language, which means that each sentence is parsed and executed by an interpreter as soon as it is submitted by the user. J, like R and other interpreted languages, has a command line interface. The user types a command on one line, strikes "Enter", and sees the result starting on the next line. The line awaiting a command has a cursor indented three spaces; the results have no indentation. This user interface greatly facilitates interaction, experimentation, and learning. The reader is encouraged to keep a J session open to run the examples below and to experiment with variations of them.

As we have seen, J uses grammatical terms for its "parts of speech". It is, after all, a language! Some of these terms have conventional mathematical equivalents, which are also perfectly appropriate and which we use interchangeably with their J counterparts. We start with nouns.

2.2.1. Nouns

Nouns are arrays of numbers, characters, or boxes. The rank of a noun is the number of its axes. J calls scalars, or 0-dimensional arrays, *atoms*; it calls vectors, or 1-dimensional arrays, *lists*; and it calls matrices, or 2-dimensional arrays, *tables*. The *items* of a noun of rank `1` or greater are its components along the first axis; each item has rank one less than the rank of the original noun. For example, the items of an array of shape `3 4 5` are matrices, or planes, of shape `4 5`.

Ordinary numbers are spelled in the usual way, except that a decimal point must always have a preceding digit, and `_` rather than `-` is used to signal a negative number (this avoids confusing the spelling *of* a number with the application of the function `-`, negation, *to* a number). The symbol `_` by itself stands for infinity, and `__` (two underscores) therefore stands for negative infinity. Complex numbers are represented as, for example, `3j4` for 3+4i. Numbers may be written in scientific notation, such as `6.022e23` or `2.998e8`, or in terms of powers of π, such as `3p4` for $3\pi^4$, or powers of *e*, as `3x4` for $3e^4$. There are other options; see **Help|Dictionary|Constants**. Numeric vectors are written by juxtaposing numbers with intervening white space; such vectors are treated as single words by J: for example, `1 2 3`.

Character atoms are single letters, spaces, digits, or other symbols enclosed in single quotes. Character vectors, or *strings*, are written by juxtaposing characters, all surrounded by a pair of single quotes; such vectors are treated as single words by J: for example, `'abc'`. To treat a quote as itself a character, it must be doubled. Characters and character vectors are displayed as entered, except with no surrounding or doubled quotes:

```
   'one year''s written premium'          Entered like this
one year's written premium                Displayed like this
```

The base library of J provides (in the z locale, searchable from any workspace; see below) ordinary names for several non-printing characters, such as `TAB`, `LF` (linefeed), and `CR` (carriage return), which are difficult to enter directly.

Boxes may contain arrays of any type. They cannot be written directly but must be constructed from their contents using J functions, such as `<` (*box*) for a single boxed array or `;` (*link*) for a vector of boxed arrays. Arrays of boxes are displayed using lines; for example

```
1;2;3 4 5                                 Entered like this
```

```
┌─┬─┬─────┐
│1│2│3 4 5│
└─┴─┴─────┘
```

Displayed like this

2.2.2 Verbs

Verbs, or functions, apply to nouns and return nouns.

A J verb must have a right argument, even if it is ignored. A function with only a right argument is said to be *monadic*, or a *monad*. A function may also have a left argument, in which case it is said to be *dyadic*, or a *dyad*. The same name may be shared by one monad and one dyad; such a name is called *ambivalent*, i.e. having both valences. Often the monad and dyad sharing the same name are related; for example, the monad may be equivalent to the dyad with a particular left argument. Monadic `%`, reciprocal, is equivalent to dyadic `%`, division, with a left argument of 1. But monadic `<` stands for box, and is not related to dyadic `<`, which stands for "less than".

Like any language, J may be extended with new functions and other objects to solve new problems. But the functions and operators supplied by the core language are the starting points for all user-defined objects, and it is they that give the language its unique flavor. The core functions of J may be classified as assignment, numeric, logical, and structural.

Assignment. Assignment is part of many examples below, and it involves only two functions, `=.` (local assignment) and `=:` (global assignment). Each might be translated "let `x` equal `y`". The left argument to either of these functions is a name defined by the user, or a list of such names separated by spaces and enclosed in quotes. A user-defined name must be spelled with letters, digits, and underscores, must begin with a letter, and normally must not contain two adjacent underscores or end with an underscore (these are reserved for locatives, names with a pointer to a specific locale). The right argument to `=.` or `=:` is any object (not necessarily a noun). For example:

```
   A=:1 2 3                    Assignment of a noun to A
   A
1 2 3
   Power=.^                    Assignment of a verb to Power
   3 Power 4
81
```

The first input line assigns a value to `A`. The second input line requests that `A` be displayed. An assignment *has* a result but suppresses its display as redundant.

A value assigned locally within a definition is available only within that definition. It does not escape into the calling environment nor seep into objects called from the definition. A value

assigned globally is available everywhere, unless shadowed within a definition by a value previously assigned locally. A value assigned on the command line, whether with `=.` or `=:` , is always global.

Multiple names may be assigned noun values in a single statement like this:

```
   'a b c'=.10 20 30
   b
20
```

The second item of the open array `y` is assigned to `b`

or like this, with boxed items (remember that `;` , link, creates a vector of boxes):

```
   'a b c'=.10;'xyz';20 30 40
   b
xyz
```

It is the opened contents of the items of `y` that are assigned

Multiple assignments may also be effected in a single line by repeated use of `=.` or `=:` :

```
   S=.+/B=.A=.3 4$i.10
   A
0 1 2 3
4 5 6 7
8 9 0 1
   S
12 15 8 11
```

The meaning of the functions `i.` , `$` , and `+/` will be explained below. For now the important things are that `A` and `B` were assigned the same value and that the result of the assignment to `B` was used as a right argument to `+/` to construct the value to be assigned to `S` .

Numeric functions. J has all the usual dyadic numeric functions, each with a monadic counterpart, and many less common but useful pairs of monadic and dyadic numeric functions. Some examples:

Name	Monad	Dyad	Remarks
`+`	Complex conjugate	Addition	J supports complex numbers
`-`	Negation	Subtraction	Monad is dyad with left argument 0
`*`	Signum	Multiplication	Signum for complex `y` is on unit circle
`%`	Reciprocal	Division	Monad is dyad with left argument `1`
`^`	Exponentiation	Power	Monad is dyad with left argument `1x1` *(e)*
`^.`	Natural logarithm	Logarithm to base `x`	Monad is dyad with left argument `1x1` *(e)*
`|`	Magnitude	Residual (`y` mod `x`)	If `y>0`, magnitude is residual with l.a. `_`
`<.`	Floor	Lesser of	Floor rounds down to nearest integer
`>.`	Ceiling	Greater of	Ceiling rounds up to nearest integer
`+.`	Real, imaginary	GCD	GCD with Boolean arguments is logical or
`*.`	Magnitude, angle	LCM	LCM with Boolean arguments is logical and
`%.`	Matrix inverse	Matrix divide	Dyad gives least squares solutions directly
`%:`	Square root	`x`-th root	Monad is dyad with left argument `2`
`!`	Factorial, gamma	Combinations	Dyad is combinations of `x` things from `y`

There are many others, including a complete set of circular and hyperbolic functions, conversions to and from different bases, derivatives and indefinite integrals of polynomials represented by their coefficients or their roots, and several additional "monads of convenience", such as `+:` (double), `-:` (halve), `*:` (square), `>:` (increment), `<:` (decrement), and `-.` (complement), which perform operations that could be done, though sometimes more awkwardly, by existing dyads. Of these, `-.` is perhaps the most useful, for, when restricted to Boolean arguments, it represents logical not.

Logical functions. The logical functions are those that return Boolean values of `0` (false) or `1` (true). These values are ordinary numbers, so that calculations dependent on a logical result may often be handled via multiplication instead of by an if-else control structure. Some of the logical functions take Boolean arguments, others (mostly comparisons) accept various other types of argument. All of the logical functions except `-.` (not) are dyads. We have already mentioned that `+.` (or) and `*.` (and) are simply GCD and LCM, respectively, applied to Boolean values. The other dyads are:

Name	Dyad	Remarks
`+:`	Nor	`x+:y` is equivalent to `-.x+.y`
`*:`	Nand	`x*:y` is equivalent to `-.x*.y`
`-:`	Match	Rank _; tests if arrays are identical; uses tolerant comparison
`=`	Equal	Rank `0`; tests element by element; uses tolerant comparison
`~:`	Not equal	Rank `0`; `x~:y` is equivalent to `-.x=y`; uses tol. comparison
`<`	Less than	Uses tolerant comparison
`<:`	Less than or equal	Uses tolerant comparison
`>`	Greater than	Uses tolerant comparison
`>:`	Greater than or equal	Uses tolerant comparison
`e.`	Membership	Returns `1` if `x` is an item of `y`, otherwise `0`

There are 16 possible Boolean dyads taking Boolean arguments; `*.` (and), `+.` (or), `*:` (nand), and `+:` (nor) are but four examples. Any of the 16 may be constructed from just `+.` and `-.`, and each has an equivalent J function, in most cases a comparison restricted to Boolean arguments. J also provides a means (the conjunction **b.**) of defining any of the Boolean dyads from its truth table.

The *comparison tolerance* is a global parameter `t` such that `x=y` returns `1` (and other comparisons behave similarly) if and only if the magnitude of the difference between `x` and `y` is smaller than `t` times the greater of the magnitudes of `x` and `y` . It is made necessary by the fact that computers represent numbers using a finite number of binary digits, limiting the precision of results. By default, `t` is `2^_44` (about `5.68e_14`). This may be changed globally using the foreign conjunction `!:` and may be changed for individual comparisons using the customize conjunction `!.` (see below for a discussion of conjunctions):

```
   (1+1e_15) = 1+1e_20                    Equal (i.e. indistinguishable) with default t
1
```

```
   (1+1e_15) (=!.1e_16) 1+1e_20      Not equal with smaller t of 1e_16
0
```

The membership function, `x e. y`, looks for matches between items of `y` and subarrays of `x` with the same rank as the items of `y`. Thus

```
   3 4 5 e. 1 3 5 7 9      Items of y have shape empty, so e. seeks atoms of x in y
1 0 1                      The first and third atoms in x are also items of y
   3 4 5 e. i.5 3          Items of y have shape 3, so e. seeks vectors of x in y
1                          The whole of x is indeed an item of y
   3 4 5 e. i.5 4          Here the items of y have shape 4, which is not the shape
0                          of a rank-1 subarray of x. So e. returns 0.
   (i.3 3) e. 1 3 5 7 9    The rank of the result is rank of x less rank of an item of y
0 1 0
1 0 1
0 1 0
```

Structural functions. We include among the structural functions those that change the rank or shape of an array, that extract elements or subarrays from an array, or that return information about the structure of an array. Some of these functions are dyads or monad – dyad pairs:

Name	Monad	Dyad	Remarks
`$`	Shape of	Assign shape	Monad reads the shape, dyad modifies it
`|.`	Reverse	Rotate `y` by `x` places	Monad is not a special case of dyad!
`|:`	Transpose	Generalized transpose	Monad reverses the order of the axes; dyad moves the axes of `y` named in `x` to the end
`,`	Ravel	Append (catenate)	Ravel (unravel?) converts any array into a vector
`;`	Raze	Link: `(<x),y` or `(<x),<y`	Raze ravels `y` and opens and assembles its elts. Link boxes `y` if and only if it is originally open
`,.`	Ravel items	Append corresp items	Monad will convert a vector into 1-column mat
`,:`	Itemize	Laminate	Both add a new first axis Monad will convert a vector into 1-row matrix
`{.`	Head	Take first `x` items of `y`	Monad takes first item, which reduces rank by 1. If `x<0`, dyad takes last `|x` items Dyad overtakes with fill items if `(|x)>#y`
`}.`	Behead	Drop first `x` items of `y`	Monad is dyad with left argument `1` If `x<0`, dyad drops last `|x` items
`{`	Catalogue	From	Monad returns all combinations of one atom each from opened items of boxed array `y` Dyad takes from `y` items indexed by atoms of `x`
`#`	Tally	Copy	Monad counts items; dyad repeats each item of `y` `x` times
`/:`	Grade up	`x` in asc. order of `y`	Monad gives a permutation of the indices
`\:`	Grade down	`x` in desc. order of `y`	Monad gives a permutation of the indices
`]`	Identity	Right identity	Useful for introducing `y` in tacit programming
`[`	Identity	Left identity	Useful for introducing `x` in tacit programming
`i.`	Integers	Index of first instance of `y` in `x`	Monadic `i.` fills an array of shape `y` with successive integers starting with `0`
`-.`		Set difference	Removes items from `x` if found as cells in `y`

Others (as structural functions) are monads only, for example:

Name	Monad	Remarks
`<`	Box	Encloses all of `y` in a box, making it an atom of type boxed
`>`	Open	Opens outer level of boxes to create items of new array, if the types agree; otherwise signals a domain error
`{:`	Tail	Last item of `y` (result has rank 1 less than the rank of `y`)
`}:`	Curtail	Drops last item of `y` (result has the same rank as `y`)
`~.`	Nub	Unique items of `y`, ordered as found in `y`
`~:`	Nub sieve	Boolean vector flagging the first instance in `y` of each item of `~.y`

2.2.3 Syntax of nouns and verbs

There is *no precedence hierarchy* for functions in J; they all apply from right to left, by which we mean that they have long right scope and, if dyadic, short left scope. While this is a departure from conventional notation, it is *much easier to remember* than a precedence hierarchy and it is a *much more practical* way of managing the large number of functions that J provides, not to mention user-defined functions, which follow exactly the same rules. It greatly reduces the need for parentheses.

```
   10*5+4*3+2
250
   10*(5+(4*(3+2)))
250
```

The first two lines here illustrate the absence of a precedence hierarchy between `*` and `+`, and the last two lines use parentheses to show the order of operation. Notice that the `*` between the `4` and the `3` has long right scope (its right argument is the result of evaluating everything to its right) but short left scope (the nearest noun expression to its left, or just the `4`).

This principle alone will allow you to decode any expression involving just verbs and nouns. Ambivalent names pose no problem; the context will always reveal how such a name is to be interpreted. When the time comes to execute the name, there either is or is not a noun to its left that is not bound to a conjunction (which is checked first by the parsing table).

Ordinary parentheses are the only symbols used in J to modify the order of operation by grouping. It would be very wasteful to use multiple symbols for the same purpose, as brackets and curly braces make four symbols available for other uses.

An expression within parentheses may evaluate to an object of any type.

2.2.4 Agreement of Arguments

A dyadic function of left rank `r` and right rank `s`, applied to arguments of rank greater than `r` or `s`, must somehow find corresponding cells on each side to work on.

If the rank of `x` is greater than `r`, then the last `r` elements of the shape of `x` determine the shape of its cells. The shape before the last `r` elements is called the *frame* of `f` as applied to `x` , or its *left frame*. Similarly for `y` and the *right frame*. If the frames match there is no problem; the function finds the cells on each side, operates on each pair of cells, and returns the result in the common frame.

If the frames do not match, but one frame is a prefix of the other, the argument with the shorter frame may be extended so that the frames match. This is done by appending additional axes and

then extending the array by repetition of its items. If neither frame is a prefix of the other, the system signals a length error.

```
   a=.i.2 4
   a
0 1 2 3
4 5 6 7
   b=.2 4$4#10 20
   b
10 10 10 10
20 20 20 20
   c=.10 20
   a+b
10 11 12 13
24 25 26 27
   a+c
10 11 12 13
24 25 26 27
```

Here `a+b` makes perfect sense: `a` and `b` have the same shape, `2 4`. The shape of `c` is just `2`, yet `a+c` is identical to `a+b`. This is because (1) the rank of the function `+` is `0`, so `+` works on individual atoms of its arguments; (2) the left frame of `a+c` is therefore `2 4`, while the right frame is just `2` ; (3) `2` is a prefix of `2 4`, so the right argument is extended to shape `2 4`, with a second axis, by repeating the elements of its first axis; and (4) in this way, `c` becomes identical to `b` and the cell-by-cell addition produces the same result as `a+b`.

```
   d=.100
   d+a
100 101 102 103
104 105 106 107
```

Here the left argument has shape empty (it is an atom), and empty is a prefix of `2 4`, so the left argument is given two additional axes and is extended by repetition to fill them, becoming a matrix of shape `2 4` with all cells equal to `100`. The left and right frames now agree.

```
   e=.100 200 300
   e+a
|length error
| e +a
```

Here the frame of `e`, the left argument, `3`, is not a prefix of `2 4`, and therefore `e` cannot be extended to an array with the shape of the frame of `a`, the right argument.

```
   a3d=.i. 3 2 4
   a3d+e
100 101 102 103
104 105 106 107

208 209 210 211
212 213 214 215

316 317 318 319
320 321 322 323
```

It all works cleanly in higher dimensions. Here `a3d` is a 3-dimensional version of `a`. The frame of the left argument to `+` is now `3 2 4`, while the frame of the right argument is `3`, a prefix of `3 2 4`. So `e` is expanded with two new axes, with its elements replicated to fill the new cells. It becomes a plane of `100`'s above a plane of `200`'s above a plane of `300`'s, and the addition can proceed normally cell by cell.

The shape of the result of a function will always be the frame followed by the shape of the result of the function as applied to each cell.

2.2.5 Fill

Somewhat similar to agreement of arguments is the use of fill values in results. For example, if `x` is a non-negative integer, `x{.y` selects the first `x` items of `y`; if `x` is negative, `x{.y` selects the last `|x` items of `y`. When `|x` is greater than the length of `y`, `{.` overtakes `y`, padding the result with *fill values*, by default `0`'s for numeric arrays, spaces for character arrays, and boxed empty vectors, called *aces*, for arrays of boxes. The fill value may be modified with the fit conjunction, `!.`, as shown in the following example; conjunctions in general are discussed in section 2.2.7 below.

```
   7{.1 2 3
1 2 3 0 0 0 0
   7 ({.!.10) 1 2 3
1 2 3 10 10 10 10
```

Some other functions that use fill values include `$` and `|.` (which by default cycle through the right argument), and `,.`. To illustrate:

```
   7 $ 1 2 3
1 2 3 1 2 3 1
   7($!.100)1 2 3
1 2 3 100 100 100 100
   3 |. 1 2 3 4 5 6 7 8 9 10
4 5 6 7 8 9 10 1 2 3
   3(|.!.0) 1 2 3 4 5 6 7 8 9 10
4 5 6 7 8 9 10 0 0 0
```

`$` fills an array of shape `x` with items from `y`, cycling through these items if needed.
But we can change the fill behavior with `!.`

`|.` shifts `y` to the left if `x>0`, rotating items in from the right
`!.` can change this to fills

Dyadic `,` , append or catenate, requires a bit more explanation. The expression `x,y` appends *items* from `y` to `x`. If one of `x` and `y` is an atom, `,` replicates it throughout an array of the shape of an item of the other. If the arrays now differ in rank, `,` adds new leading dimensions to the one with smaller rank. If the items of the arrays now differ in shape, `,` pads either or both with fill values, by default `0` 's, spaces, or aces. This procedure generalizes to higher dimensions the idea of appending items to a vector.

```
   1 2 3,40 50
1 2 3 40 50
   A=.i.2 5
   A
0 1 2 3 4
5 6 7 8 9
   A,10
 0  1  2  3  4
 5  6  7  8  9
10 10 10 10 10
   A,10 20
 0  1 2 3 4
 5  6 7 8 9
10 20 0 0 0

   B=.i.2 3 4
   {:B,10 20
10 20 0 0
 0  0 0 0
 0  0 0 0
```

The simple case of two vectors is by far the most common situation. Items are atoms, and `,` appends items from the right to the left arg. Here `A` is a matrix.

Each item of `A` has shape `5`.

The scalar `10` is extended by replication to the shape `5` of an item of `A`, then given a new leading axis to become a matrix of shape `1 5` . This matrix is then appended to the matrix `A` .

The vector `10 20` is given a new leading axis to become a matrix of shape `1 2` , the only item of which is padded with fills to the shape of an item of `A`. It now has shape `1 5` and may be appended to `A` .

Here `B` is an array of rank `3`
What is the last item of `B,10 20` ?
It is the only item of the result of appending two leading axes to `10 20` to make it an array of shape `1 1 2` , then padding with fills to give it shape `1 3 4` . The *item* has shape `3 4` as shown here.

The need for fills with `,` does not arise when each of **`x`** and **`y`** is an atom or a vector. Dyadic `,.` (catenate corresponding items, or *stitch*), which is defined in terms of `,` , may need fills when the rank of `x` or of `y` is greater than `2`; and dyadic `,:` (laminate) may need fills when both arguments have rank greater than `0`.

2.2.6 Adverbs

An adverb modifies a function, or occasionally a noun, to produce a new function. The adverb is written to the right of the object it modifies.

The symbol `/` (monadic *insert*, dyadic *table*) is a good example of an adverb. It modifies a dyadic function `f` to produce a new function, which is ambivalent. The monad `f/` inserts `f` between each pair of items in the argument of `f/` . So `+/1 2 3 4` becomes `1+2+3+4` ; `+/` is summation. Likewise `*/y` gives the product, `<./y` gives the minimum, and >./`y` gives the maximum, over the items of `y` . If `y` is Boolean, `*./y` gives `1` if and only if all its items are `1` (or if it is empty; J is very

logical in the treatment of edge conditions), and `+./y` gives `1` if and only if at least one element of `y` is `1`.

The dyad `f/` creates a table of `f` applied to all combinations of one item each from `x` and `y`:

```
   2 3*/4 5 6
 8 10 12
12 15 18
```

Another common adverb is `}` (*amend*), which lets us insert items from `x` into particular positions in `y`. This adverb modifies a noun `m` rather than a function, but it still produces a function as its result. The noun `m` gives the desired index or indices in `y`. Yes, arrays have indices, even though most functions never need to reference them, and `}` uses the same indices as `{`. The new function `m}` inserts *its* left argument into the cell or cells of *its* right argument as specified by the noun `m`:

```
   A=.'abcde'
   B=.'-' 1 3}A
   A
abcde
   B
a-c-e
```

The line defining `B` inserted `'-'` into positions 1 and 3 of `A` (J has index origin 0). The operation `}` has no effect on `A` itself; it simply creates a new array, on the fly, which we chose to assign to `B`. If we wanted to change `A`, we could have assigned the new array to `A`. If we wanted to process the result further before assigning it, we could do so, and the original amended array, created on the fly, would vanish on the fly. So "amending" properly takes place only if the result of `m}` is reassigned to the original name.

The adverb `~` is particularly useful in tacit definitions. It applies to a dyadic function `f`, but its result is ambivalent. As a monad, `f~ y`, called *reflexive*, gives `y f y`; as a dyad, `x f~ y`, called *passive*, gives `y f x` (commutation). We are no longer obliged to put the divisor after the division symbol, or the subtrahend after the subtraction symbol, or the exponent after the base, if the reverse is more convenient. One common use of `~` is to put a vector `y` in order. As noted above, the function `/:`, used dyadically, puts its left argument in the order determined by its right. If we want to put `y` in an order determined by itself, we need only write `/:~y`.

Subarray adverbs. One useful family of adverbs consists of those that modify a monadic function `f` to apply to various subarrays of its argument. The result may be monadic or dyadic; if it is dyadic, its left argument specifies precisely which subarrays are to be used:

Name	Adverb	Result	Behavior of result function
`/`	Insert	Monad	Inserts a copy of `f` between successive items of `y` (example above)
`\`	Prefix	Monad	Applies `f` to successive prefixes of `y` (ending with `y` itself)
`\`	Infix	Dyad	Applies `f` to successive "infixes" (interior segments) of length `x`
`/.`	Oblique	Monad	Applies `f` to successive subdiagonals of a matrix `y`
`/.`	Key	Dyad	Applies `f` to `y` grouped by the unique values of `x`
`\.`	Suffix	Monad	Applies `f` to successive suffixes of `y` (starting from `y` itself)
`\.`	Outfix	Dyad	Applies `y` to the complements of successive infixes of length `x`

```
   i.5
0 1 2 3 4
   +/\i.5
0 1 3 6 10
   <\.i.5
```

0 1 2 3 4	1 2 3 4	2 3 4	3 4	4

2.2.7 Conjunctions

A conjunction connects two verbs, or a verb and a noun, or two nouns, to produce a new verb (or, rarely, a new adverb). The conjunction is written between the objects it modifies. We could have called an adverb a "monadic conjunction, except that we do not want to become confused by the fact that the *result* of an adverb or conjunction, which is a function, may itself be either monadic or dyadic.

Rank. The conjunction `"` (*rank*) is in a class by itself, because it embodies so much of what makes J unique. It applies to any function `f` and defines or redefines the rank of the cells on which `f` works. A function may have three ranks: monadic, left, and right. The rank conjunction takes a noun right argument, which may be a vector of length `3`, a vector of length `2` (taken to be the dyadic ranks, the monadic rank then being set equal to the right dyadic rank), or a scalar or vector of length `1` (all three ranks the same).

If an argument `y` has, say, rank `4`, with shape, say, `2 3 4 5`, and a function `f` has rank `1` and produces a result of rank `0` (say, the Euclidean length of a vector), then, when `f` is applied to `y`, it works on each cell of rank `1` (a vector of length `5`), producing a scalar result, and then assembles all these results into a final result of shape `2 3 4`. This is the right frame, `2 3 4`, followed by the shape of `f` as applied to each cell, which is empty.

Our `f`, which returns the square root of the sum of the items of a vector `y`, might be defined by

```
   f=.%:@:(+/)@:*:"1
```

The `*:` does the squaring, the `+/`does the summing, and the `%:` does the square root. The `"1` at the end sets the rank of `f` to `1`, no matter what the shape of its argument. The appearances of the conjunction `@:`, discussed below, successively compose the three functions.

```
   y=.i.3 4
   y
0 1 2 3
4 5 6 7
8 9 10 11
   f 0{y
3.741657387
   f 1{y
11.22497216
   f 2{y
19.13112647
   f y
3.741657387 11.22497216 19.13112647
```

So the result of `f` applied to the matrix argument `y` is indeed a vector of the results of `f` applied to each row.

If `f` had been defined without the `"1`, it would by default have had rank _ (no limit). The `*:` (square) and `%:` (square root) have rank `0` and would apply as usual to each *atom* of their arguments, but the `+/` has rank _, and would insert a `+` between each *item* of *its* argument. Remember that the items of an array are its components, along the leading dimension, of rank one less than the rank of the array. Here the items are vectors of length `4`. Now `+`, with rank `0` on both left and right, operates on each pair of vectors element by element, so `+/` in effect sums *down the columns*. Using `g` in place of `f` without the `"1`, we obtain:

```
   g=.%:@:(+/)@:*:
   g 0{y
3.741657387
   g y
8.94427191 10.34408043 11.83215957 13.37908816
   g 0{"1 y
8.94427191
```

When `g` is applied to a row of `y`, the result is the same as before, because each row has rank `1`. But when `g` is applied to all of `y`, the result is different! It gives the Euclidean lengths of the *column* vectors of `i.3 4` . If we reset the rank of `g` to `1` using `"1`, we get the same result as with `f` :

```
   g"1 y
3.741657387 11.22497216 19.13112647
```

If `y` had had rank `3`, the result would have been a matrix each element of which contained the result of `f` applied to the corresponding rank `1` cell of `y`.

```
   y=.i.2 3 4
   f y
3.741657387 11.22497216 19.13112647
27.09243437 35.07135583 43.05810028
```

As one more example, imagine we have a 4-dimensional array `y` giving paid losses for a group by company, line, accident year, and payment year. As we shall see below, the monad `f =: +/@,` sums over all the elements of an argument whatever its shape (that is, the rank of `f` is _). So `f y` gives paid losses for all companies in the group. Reducing the rank, `f"3 y` gives ITD paid losses by company; `f"2 y` gives paid losses by line within company; `f"1 y` gives paid losses by accident year within line within company; and `f"0 y` gives `y` unchanged (summing over an atom doesn't change it). We can permute the axes of y using `|:` if we wish the nesting to be in a different order.

The rank conjunction `"` with noun left argument gives a constant function with that value and with rank(s) given by the right argument. The rank conjunction may also be used to assign the rank(s) *of* a function right argument to the function or constant specified by the left argument.

Cut. The conjunction `;.` (cut) takes a noun right argument and, usually, a function left argument. Cut is similar to the subarray family of adverbs. It can specify very general subarrays, and its right argument is a code that controls the choice of subarray.

A simple use of cut is to parse data inherited from other systems as character files. For example, `UnD1`, one of our library functions, which might be called "undelimit", separates a character vector `y` into subvectors delimited by a character given by `x` . It does this by first appending the delimiter to the end of `y` if not already there, and then applying the phrase `<;._2`, which means "box the elements between, but not including, instances of the last element of `y`".

```
   '|' UnD1 'abc|defgh||ijk|lmnop'
┌───┬─────┬┬───┬─────┐
│abc│defgh│├┤ijk│lmnop│
└───┴─────┴┴───┴─────┘
```

Composition. A large family of conjunctions lets us compose monadic and dyadic functions in various ways. These include:

Name	Conjunction	Arguments	Result	Behavior of result function
`@`	Atop	M M	Monad	`f@g y` equiv. `f g y`, with rank of `g`
`@`	Atop	M D	Dyad	`x f@g y` equiv. `f x g y`, with rank of `g`
`@:`	At	M M	Monad	Same as `@` except rank of result is _
`@:`	At	M D	Dyad	Same as `@` except rank of result is _
`&`	Bond	N D or D N	Monad	`n&f y` equiv. `n f y`; `f&n y` equiv. `y f n`
`&`	Compose	M M	Monad	`f&g y` equiv. `f g y`, with rank of `g`.
`&`	Compose	D M	Dyad	`x f&g y` eq. `(g x) f g y`, with rank of `g`
`&:`	Appose	M M	Monad	Same as `&` except rank of result is _
`&:`	Appose	D M	Dyad	Same as `&` except rank of result is _
`&.`	Under	M M	Monad	Same as `&` except applies inverse of `g` to result
`&.`	Under	D M	Dyad	Same as `&` except applies inverse of `g` to result
`&.:`	Under	M M	Monad	Same as `&.` except rank of result is _
`&.:`	Under	D M	Dyad	Same as `&.` except rank of result is _
`^:`	Power	M N	Monad	Compose `f` with itself `n` times
`^:`	Power	D N	Dyad	Compose `x&f` with itself `n` times

Of all these conjunctions, only monadic `@` and `&` (which are equivalent) have a corresponding symbol, ∘, in conventional notation. We discuss function composition in detail in Section 2.3.

Numeric operators. Several conjunctions are numeric in both their arguments and their result function:

Name	Conjunction	Result	Behavior of result function
`.`	Determinant	Monad	`-/ .*` gives determinant of `y`, `+/ .*` gives permanent; this is generalized for other functions, using expansion by minors
`.`	Dot product	Dyad	`x +/ . * y` gives matrix product of `x` and `y`; this is generalized for other functions
`..`	Even	Monad	`((f y)+f-y)%2` when `g` is negation; generalizes to other `g`
`.:`	Odd	Monad	`((f y)-f-y)%2` when `g` is negation; generalizes to other `g`
`D.`	Derivative	Monad	`f D. n` gives *n*th derivative, or partial derivatives, of `f`
`d.`	derivative	Monad	`f d. n` gives *n*th derivative of `f` (assumed to have rank 0)

For actuaries, the most commonly used of the above conjunctions will probably be the dot product. But the derivatives are also interesting. For example:

```
   f=.5 4 3&p.
   g=.f d. 1
   f 2 3 4
25 44 69
   g 2 3 4
16 22 28
```

The first line defines `f` as the polynomial $f(x)=5 + 4x + 3x^2$. The second line defines `g` as the first derivative of `f` , i.e. as $4 + 6x$. The next four lines show that `f` and `g` behave as expected when `x` is `2`, `3`, or `4`

```
   f=.(5 4 3&p.@:{.)+0 0 0 1&p.@:{:
   f 2 3
52
   (f D. 1)2 3
16.00000061 27.00000271
   ((5 4 3&p.)D.1) 2
16
   ((0 0 0 1&p.)D.1) 3
12
```

Here the first line is one way of defining `f` as the polynomial $f(x,y) = 5 + 4x + 3x^2 + y^3$, when `f` is a monad with argument a vector of length `2` giving (x, y). The second and third lines confirm that `f` behaves as expected when x is 2 and y is 3. The next two lines give the partial derivatives of f with respect to x and y at (2, 3). The last four lines show that `D.` (or `d.`) handles single-variable polynomials (rank of `f` is `0`) more precisely than multi-variable polynomials (rank of `f` is `1`), probably by using direct rules rather than numeric approximations for differentiation.

System operators. Yet another class of conjunctions contains those that modify the behavior of the interpreter, or that return information about the properties of functions, or that communicate with files or other system resources or return information about system performance. Most of their arguments are nouns interpreted as codes to select which member of a set of related operations is intended. This creates whole families of functions. The largest of these families is provided by the *foreign conjunction*, `!:`, which handles queries and commands foreign to the J language proper. Among these the most immediately useful are probably `1!:y` (files), `9!:y` (global parameters), and `13!:y` (debugging).

The file functions include `1!:1` (read), `1!:2` (write), `1!:3` (append), `1!:11` (indexed read), `1!;12` (indexed write), and many others.

Global parameters include the print precision, which the reader will likely wish to increase above its default value of 6 digits. This is done by executing, e.g., `(9!:11) 10` to set it to 10 digits.

The debugger enables the user to follow execution line by line, to set stop points, and to cause the debugging screen to open, without exiting the function, on error. Its behavior is controlled by the `9!:13` family, though it is usually easier to use the cover functions that J creates in the **z** locale, such as `dbss` (set stops).

The reader should consult **Help|Foreign Conjunction** for detailed information.

Name	Conjunction	Arguments	Result	Behavior of result function
`!.`	Customize	M N, D N	M, D	Most common use is to change the fill value in `{.` , `|.` , `$` , `,` ,etc. Other uses include changing the comparison tolerance for `=` , `<` , `<:` , etc.
`!:`	Foreign	N N	M D	Creates a large collection of functions relating, among other things, to files, operating system, resources and performance, display formatting, querying or setting global parameters, managing names, and data representation and conversions.
`:`	Explicit	N N	M D	Permits explicit definition (one or more statements referencing argument(s), defining the result as that of the last expression evaluated)
`:`	Monad-Dyad	N N	M`:`D	Separates monadic and dyadic parts of a definition, either tacit or explicit

2.2.8 Syntax of adverbs and conjunctions

There is no precedence hierarchy among adverbs and conjunctions. These operators apply from left to right, by which we mean they have long left scope and short right scope. This is the opposite of the direction of reading functions applied to nouns. It can never lead to ambiguity.

When applying the various composition conjunctions, parentheses may occasionally be needed to confine the conjunctions to the desired functions. For example:

```
   sum=.+/
   f=.,@:sum"2
   g=.,@:(sum"2)
   y=.i.2 3 4
   f y
12 15 18 21
48 51 54 57
   g y
12 15 18 21 48 51 54 57
```

The function `f` (sums-down-columns and then ravels) each plane of `y`. The function `g` (sums-down-columns each plane) and then ravels the result. The difference is the scope of `"`.

Parentheses may also be necessary to avoid confusing a numeric argument to an adverb or conjunction with part of a neighboring noun.

```
   9!:11 10      An attempt to set the print precision
|rank error     It doesn't work! The right argument of 9!: must have rank 0. The 10 is
| 9 !:11 10     meant to be the right argument of the result of 9!:11, not of !: itself.
   (9!:11)10     Now it works.
   100%7
14.28571429      10 digits are displayed.
```

2.2.9 Hooks and Forks

Hooks and forks are constructs, unique to J, that permit functions to be combined using other functions as intermediaries, much as *f*+*g*, *f***g*, min(*f*,*g*), and so forth, do in conventional notation. They are notated by the juxtaposition of two function names (a hook) or three (a fork), which must be surrounded by parentheses in explicit definitions, and which have the following meanings:

Notation	Arguments	Name	Result	Behavior of result function
`(f g)`	D M	Hook	M	`(f g) y` is equivalent to `y f g y`
`(f g)`	D M	Hook	D	`x (f g) y` is equivalent to `x f g y`
`(f g h)`	M D M	Fork	M	`(f g h) y` is equivalent to `(f y) g h y`
`(f g h)`	D D D	Fork	D	`x (f g h) y` is equiv. to `(x f y) g x h y`
`(n g h)`	N D M	Fork	M	`(n g h) y` is equivalent to `n g h y`
`(n g h)`	N D D	Fork	D	`x (n g h) y` is equivalent to `n g x h y`
`([: g h)`	M M	Fork	M	`([: g h) y` is equivalent to `g h y`
`([: g h)`	M D	Fork	D	`x ([: g h) y` is equivalent to `g x h y`

The last two are called *capped forks*, and use the special "function" `[:` (cap) which suppresses any left argument and has only this single application.

Obviously some hooks and forks replicate the functionality of some conjunctions. What is interesting is that forks simplify tacit programming by allowing a *train* of function names, parsed into forks, from right to left in overlapping threes, possibly with a final hook. Parentheses are usually not needed in such constructs, unless we want a function, other than the right-most one, itself to be defined by a train. We give a fuller discussion of function composition, including the role of hooks and forks, in a later section.

Note that `(f+g)`, `(f*g)`, `(f<.g`), etc. replicate *f*+*g*, *f***g*, min(*f*,*g*), etc. of conventional notation.

2.3 Defining Objects

The essence of all programming is extending a language with new objects to accomplish new tasks. Either the language itself, or its development environment, or both, must provide tools for defining the new objects. In J these tools are *explicit definition* as provided by the conjunction `:` , and *tacit definition*, in which the new function is defined directly from existing functions, adverbs, conjunctions, hooks, and forks.

Explicit definitions. Any part of speech may be defined explicitly. The left argument of `:` is `0` to define a noun, `1` for an adverb, `2` for a conjunction, `3` for a verb, `4` for a verb required to be dyadic, or `13` to convert an explicit one-line verb definition into tacit form if feasible.

Within an explicit definition, the names `x` and `y` are reserved for the left and right arguments of a function, the name `u` for a left argument of an adverb or conjunction, the name `v` for the right argument of a conjunction, the name `m` for a noun left argument of an adverb or conjunction, and the name `n` for a noun right argument of a conjunction.

The script **act.ijs** provides hundreds of definitions to study. Here are a few simple examples.

One of our utility functions, `Subst`, substitutes the second item of `x` for each instance of the first item of `x` found in `y`.

```
   Subst=:4 :'({:x)(I.y e.{.x)}y'          NB. Explicit definition as dyad
   0 _ Subst i.5
_ 1 2 3 4                                  NB. Application to numeric vectors
   'aA' Subst 'anna eats oats'
AnnA eAts oAts                             NB. Application to character strings
```

Another function, `SSpV` replaces characters in the vector `y`, also found in `x`, with spaces:

```
   SSpV=:4 : ''' ''(I.y e. x)}y'
   'abc',TAB,TAB,'def',TAB,TAB,'ghi',LF,'jkl'
Abc     def     ghi
jkl
   (TAB,LF) SSpV 'abc',TAB,TAB,'def',TAB,TAB,'ghi',LF,'jkl'
abc  def  ghi jkl
```

Many explicit definitions require, or are easier to read with, more than one line. In this case we use, for the right argument of the conjunction `:` , the special code `0`, which means "take the text of the definition from all the following lines before the first line with nothing but a right parenthesis". A contrived example:

```
   f=: 4 : 0"0  NB. Sums of first x powers of y, starting with 1
p=.1+i.x                       NB. Vector of powers
+/y^p                          NB. Sum of powers
)
   5 f 3 4 5
363 1364 3905
```

The first four lines are the definition. This didn't really require multiple lines, but it's just an example. It illustrates the use of comments to document the purpose of the whole function and of some of its logic, the use of a local variable, `p`, defined within the function, and the assignment of a specific rank (here `0`) to the function. It is necessary to specify rank `0` lest the internal application of `^` to the array `y` and the vector `p` produce, on occasion, a length error.

Tacit definitions. We have already seen some examples of tacit definitions. They occupy a single line of code; they make no reference by name to their arguments; they do not define any local variables; they use conjunctions, hooks, and forks to compose functions. A tacitly-defined version of our function `f` above might be

```
   ft=.([:+/]^1+[:i.[)"0
   5 ft 3 4 5
363 1364 3905
```

This example reveals the simplicity and elegance of tacit programming with successive forks. If parenthesized to bring out the order of execution, `ft` would be `([:+/(]^(1+([:i.[))))"0`. Note the use of the identity functions `]` and `[` as stand-ins for `y` and `x`, and note also that the first fork and last forks are capped to accommodate the monads `i.` and `+/`, and that the second fork has a noun for its left argument, where it is permitted for convenience in lieu of a constant function.

J provides an automatic way to convert one-line explicit definitions to tacit form, where feasible; this is the conjunction `:` with left argument `13`. For example,

```
   13 : '+/y^1+i.x'"0
([: +/ ] ^ 1 + [: i. [)"0
```

We write our function `f` on one line and apply `13 :` The result is our `ft` with some additional white space

We may be able to shorten a tacit definition further by using other tacit definitions within it, especially when we know the circumstances under which the function will be called. For example, one of our library functions, `SDiag`, flags with `1`'s the `x`'th subdiagonal of a matrix of shape `y` (thereby, for example, marking cells `x` calendar years after the start of a lagged loss triangle). A simple definition might be `13 : 'x=(i.{.y)+/i.{:y'`, producing (without the extra white space) the tacit definition `[=([:i.[:{.])+/[:i.[:{:]`. But we know that this function will always be called with `y` of length `2`, so we can simplify it to `[=[:(+/&:i.)/]`, which inserts the tacit `(+/&:i.)` between the two items of `y`. This is our library function `SDiag`.

Of course the next natural question is whether `SDiag` works with lag arrays of dimension other than `2`. As written, it does not, but `[=[:>[:+/&.>/[:<@:i."0]` will do the trick. This sets up a list of boxed index vectors before combining them into a multidimensional table, thus avoiding the problem of applying `i.` to ever-higher-dimensional arrays in the successive insertions. In the library we have given this function the name `SDiagX` (extended `SDiag`).

2.4 More about Composition

Here we take up the subject of function composition in more detail.

We are all familiar with the usual composition of functions: if f maps X into Y and g maps Y into Z then the function h defined by $h(x) = g(f(x))$ maps X into Z. We usually write this as $h = g \circ f$. This expression is actually an example of tacit programming in conventional notation.

In J there are many more possibilities.

First, the definition of $g \circ f$ assumes that f and g are monadic, although we can get around this limitation a little bit by, e.g., letting the domain of f be $\boldsymbol{R} \times \boldsymbol{R}$. J goes further and provides efficient means of composing all combinations of monadic and dyadic f and g.

Second, $g \circ f$ requires that the range of f be a subset of the domain of g. This is equally true in J. But J functions have rank. For example, `+` is defined in terms of its operation on objects of rank `0`, even though its domain, in the universe of all J objects, includes numeric arrays of any rank. So J makes the notion of composition more precise by considering how the rank of the composed function relates to the ranks of f and g.

Let's look at the possibilities. In the tables below, those kinds of composition for which J provides primitive symbols or constructs, such as single hooks or forks, are printed in blue; those which must be built up using other symbols are in green. Even these are usually very concise.

Balanced compositions. If we put the functions and their arguments in a tree structure, some of the cases are balanced, with all leaves at the same distance from the root. J provides primitive symbols for all these cases:

```
g          g          g
f        f   f        f
y        x   y      x   y
```

Valence			Definition of h	Notation	
f	g	h		Rank(h) = Rank(f)	Rank(h) = _
M	M	M	`g f y`	`g&f y` `g@f y`	`g&:f y` `g@:f y`
M	D	D	`(f x) g f y`	`x g&f y`	`x g&:f y`
D	M	D	`g x f y`	`x g@f y`	`x g@:f y`

Thus J replaces the conventional symbol ∘ with four different words, `&`, `@`, `&:`, and `@:`, and makes each apply to either the "both monadic" case or the "one monadic and one dyadic" case. This covers the great majority of instances of function composition in practical programming. Note that the all-monadic uses of `&` and `@`, and of `&:` and `@:`, are identical, and that the latter pair cover the one case that might properly be written $g \circ f$ in conventional notation.

Unbalanced compositions. J can also handle unbalanced compositions, where not all leaves are the same distance from the root. Situations where `f` is monadic and `g` dyadic, but the functions are only composed on one side, look like hooks, and indeed one of them, when the result of `f` becomes the right argument of `g`, is the J hook `(g f)`. When the result of `g` becomes the left argument of `f`, the simplest expression for the left-argument case applies the commutation adverb `~` twice. As always, there are multiple ways of doing the same thing; we also show one using forks.

```
  g                g
x   f              f   y
    y              x
```

Valence			Definition of `h`	Notation	
f	g	h		Rank(`h`) = Rank(`f`)	Rank(`h`) is `_`
M	D	D	`x g f y`	`x (g f)"f y`	`x (g f) y`
M	D	D	`(f x) g y`	`x (g~ f)~"f y` `x (] g~ [:f[)"f y`	`x (g~ f)~ y` `x (] g~ [:f[) y`

Hooks and forks have infinite rank so we use the rank conjunction `"` if we want their rank to be that of `f`. To see what that rank is, we can query it by executing `f b. 0`.

Replicated arguments. J can also handle the special cases where `g` is dyadic and `f` monadic, or vice-versa, but there is only one distinct argument so the composed function `h` is monadic:

```
  g                g                g                g
f   f            y   f            f   y            f
y   y                y            y                y   y
```

Valence			Definition of `h`	Notation	
f	g	h		Rank(`h`) = Rank(`f`)	Rank(`h`) is `_`
M	D	M	`(f y) g f y`	`g&f~ y`	`g&:f~ y`
M	D	M	`y g f y`	`(g f)"f y`	`(g f) y`
M	D	M	`(f y) g y`	`(f g ])"f y`	`(f g ]) y`
D	M	M	`g y f y`	`g@(f~) y`	`g@:(f~)y`

Finally, J can handle cases where both f and g are dyadic, so that the trees necessarily involve repetition of one or both of the arguments x and y, for example:

```
   g          g          g          …          g
 f   f      x   f      f   y        …        f   y
x y x y        x y    x y          …        y y
```

Valence				Notation	
f	g	h	Definition of h	Rank(h) = Rank(f)	Rank(h) is _
D	D	D	`(x f y) g x f y`	`x (f g f)"f y`	`x (f g f) y`
D	D	D	`x g x f y`	`x ([ g f)"f y`	`x ([ g f) y`
D	D	D	`(x f y) g y`	`x (f g ])"f y`	`x (f g ]) y`
…	…	…	…	…	…
D	D	M	`(y f y) g y`	`(f g ])"f~ y`	`(f g ])~ y`

with appropriate adjustment to the rank of `[` or `]` if the left and right ranks of `g` differ.

Forks. The above tables show that J can do all forms of composition of two functions either directly with a single word or indirectly with a concise expression. But there's more! J also has ways of composing *three* functions. Some of these are simply chains linked together with `@`, `@:`, etc., but others are more interesting. The fork `(f h g)` represents the diagrams

```
   h                    h
 f   g      and       f   g
 y   y              x y x y
```

i.e. `(f y) h g y` and `(x f y) h x g y`.

As described in the section on conjunctions, the left element in a fork may be replaced by a constant, which will be interpreted as a constant function, or it may be replaced with a placeholding function "cap", notated `[:`, which allows the middle function to be applied monadically. Thus the fork `([: h g)` represents

```
 h          and        h
 g                     g
 y                   x   y
```

i.e. `h g y` and `h x g y`, so a capped fork produces the same result as `@:`. Capped forks are convenient because they simplify the writing of long tacit functions where a monad terminates or interrupts a series of dyads.

Trains. Single hooks and forks can be very useful: an example of such a hook is `(%+/)`, which translates to `y % +/y` and returns the vector `y` normalized to total `1`, and an example of a fork is the classic `(+/%#)`, which translates into `(+/y) % #y` and gives the arithmetic mean(s) of `y`

across its first dimension. But hooks and forks are even more powerful when combined to produce more complicated tacit functions. Any isolated sequence, or *train*, of *n* functions, with $n > 1$, may be parsed into a sequence of $(n-1)/2$ forks, if *n* is odd, or $(n-2)/2$ forks followed by one hook, if *n* is even. For example:

```
   (;{:%~{.,}.-}:)1 3 20 40 50
┌──────────┬──────────────────┐
│1 3 20 40 50│0.02 0.04 0.34 0.4 0.2│
└──────────┴──────────────────┘
```

This snippet from a screen session takes a vector of cumulative loss payments and converts it into lag factors normalized to total 1.00, then displays both the payments and the lag factors. The whole function is a sequence of nine primitive J operations, `; {: % ~ {. , }. - }:`, which may be broken into the following eight functions:

```
;  {:  %~  {. ,  }. -  }:
```

We treat `%~` as a unit since it is the verb `%` modified by the adverb `~`.

This expression may in turn be broken into three forks and a hook, as follows:

```
(; ({: %~ ({. , (}.-}:))))
```

The innermost fork is the three words on the right of the original expression, `}.-}:`, which takes first differences of the vector `y`, obtaining incremental losses.

The next fork appears when we pull in the next two words to the left; it appends to the beginning of the vector the first element of the original `y`. Equivalently, we could have put a `0` before `y` when taking first differences, to acknowledge that the first monthly lag started with zero cumulative losses.

The third fork divides its right side (everything so far) by its left (the last element of `y`), thereby normalizing the result (since we started with cumulative losses, the last element is the sum of all the incremental losses). Note that the `~` adverb commutes the usual arguments of `%`.

Finally we are left with the single word `;` and everything to its right, which has now been reduced to a single function. So there are two function expressions and a single argument `y`; this is a monadic hook. It is equivalent to `y;<`(everything so far)`y`, which means make a list of two boxes one containing the original `y` and the other the vector of lag factors. This is the result shown.

In conclusion, J facilitates tacit programming by having simple ways of expressing all forms of composition of two, and many forms of composition of three, functions.

As a parting example, here is how you might use some of the above ideas to do simple dollar-weighted loss development on a cumulative date x date loss triangle. We define four little functions:

```
   lag=:([:i.#)|."0 1 ]  NB. Converts date x date to date x lag
   num=:+/@:(}."1)@:lag  NB. Numerators of development factors
   den=:+/@:lag@:(}:"1)  NB. Denominators of development factors
   ldf=:num % den        NB. Average development factors
```

The first function, `lag`, is a fork with elements `([:i.#)` , `|."0 1` , and `]` . The leftmost of these elements is a capped fork; it gives the number of years by which each row of the triangle must be lagged. The second element does the lagging; it is the rotate function `|.`, modified by the rank operator `"` to apply to scalars from its left argument (the amounts of shift) and vectors from its right argument (rows of the triangle). The third element is the right identity function `]`, which in effect says "do the shifting on the right argument of the fork". Since the fork is used monadically it has only a right argument.

The second function, `num`, is a chain of three functions composed with `@:`. It says, "lag the triangle, then drop its first column, then sum down its columns". The rank operator tells `}.` to work on cells of rank `1`, or vectors, which are the rows of the triangle, and dropping the first element of each row amounts to dropping the first column of the whole triangle. We do this because the first element of each row is not the numerator of any of the development factors.

The third function, `den`, is also a chain of three functions composed with `@:`. It says, "drop the last column, then lag the resulting triangle, then sum down the columns". Here we drop the last column of the original triangle since its cells would fall on the latest diagonal of the lag triangle. They are not matched with any numerator cells and should not participate in the denominator.

The fourth function, `ldf`, is a simple fork. Voilà! Development factors. We tri it:

```
   tri=:4 4$3 4 5 6 0 7 8 9 0 0 10 11 0 0 0 12
   tri
3 4  5  6
0 7  8  9
0 0 10 11
0 0  0 12
   lag tri
 3  4 5 6
 7  8 9 0
10 11 0 0
12  0 0 0
   num tri
23 14 6
   den tri
20 12 5
   ldf tri
1.15 1.16667 1.2
```

These simple definitions presume that their right argument is a square matrix with zeros below its main diagonal – the most common situation with date x date loss triangles. But they don't impose any limits on the size of the triangle; it could just as well be a 120 x 120 monthly loss triangle imported from Excel as the 4 x 4 triangle created for this example.

Tacit programming is most suitable for situations, like this, with only a few variables. If we wanted to introduce control parameters, such as depth and type of averaging, the additional variables and the need for conditional execution would make explicit definition more convenient.

2.5 Control Structures

J supports a full complement of control structures, which it likens to punctuation. They mark the beginning and end of blocks of code to be executed under specific conditions, or to test for these conditions. In the following table, `A`, `A1`, etc., represent blocks to be evaluated to control the conditional execution; `B`, `B1`, etc., represent blocks to be conditionally executed.

Control structure	Description
`if. A do. B end.`	If-then, executing `B` if and only if the first atom of the result of the last statement of `A` is nonzero, or the result is empty
`if. A do. B else. B1 end.`	If-then-else, i.e. with an alternative path

`if. A do. B`
`elseif. A1 do. B1`
`elseif. A2 do. B2`
…
`elseif. 1 do. Bn end.`
	If-then with multiple else conditions. The last one is an optional but convenient way of accommodating a block `Bn` to be executed if all else fails

`for. A do. B end.`	Executes `B` once for each item of `A`.
`for_item. A do. B end.`	Executes `B` once for each item of `A` . The variables `item` and `item_index` hold the active item and its index, for reference in `B`; "`item`" may be any name.
`while. A do. B end.`	Executes `B` as long as the first atom of the result of `A` is nonzero, or the result is empty.
`whilst. A do. B end.`	Same as `while.` except always executes `B` once.

`select. A`
`case. A1 do. B1`
`fcase. A2 do. B2`
`case. A3 do. B3`
…
`case. An do. Bn end.`
	Evaluates `A` and selects the first case that matches the result, executing the corresponding `B` statement. Terminates if the match was with `case.`, continues to look for additional matches if with `fcase.`

`break.`	Exits a `for.`, `while.`, or `whilst.` loop.
`continue.`	Returns execution in a `for.`, `while.`, or `whilst.` loop to the top for next iteration.
`return.`	Exits an object defined explicitly.
`throw.`	Exits a function downstream from a `try.` to the nearest `catcht.` if any, otherwise to interactive execution with an error message.

`try. B catch. B1 end.`
`try. B catcht. B2 end.`
`try. B catchd. B3 end.`
(etc)
	Exits block `B` on error, resuming at `catch.`
	Exits `B` on throw., resuming at `catcht.`
	Exits `B` on error, resuming at `catchd.` *if not debugging.*
	A single try may have more than one kind of catch.

`assert. A`	Signals an error if the result of A is not all 1's. to check that expected conditions are satisfied.
`goto_name.` `label_name.`	A control structure in many languages; out of favor for a tendency to produce unclear logic.

A control structure may be entered on a single line or spread over multiple lines.

You will probably use the `if.` family most frequently, and the remaining structures much less often. It is remarkable how few `for.`, `while.`, or `whilst.` loops J programs require. Even the `if.` constructs can often be avoided by building Boolean logic into ordinary calculations.

3. LIBRARY OF ACTUARIAL MODELS

The accompanying library of actuarial models is a J script, in other words an ordinary text file of J code. It may be opened by the J editor, or copied and pasted into the editor, then loaded into a workspace by the menu items **Run|Load Script**. The author hopes that it will serve as a starting point for practicing actuaries to build upon as they program tools to perform their own tasks. Many of these tasks may be specific to a particular client: for example, converting files extracted from the client's database into our intermediate file format or creating top-level reports on various statistics of interest to the client's management. Other possibilities include linking work in J to other applications such as OpenBUGS (which does Markov Chain Monte Carlo Bayesian estimation) or to statistical models in R, either directly through a DCOM link or indirectly by pasting text output from J into the R console.

The library does not provide equal representation to all actuarial specialties, but instead reflects the author's experience. It covers some of the same ground as a collection of APL programs the author wrote over the years for casualty loss reserving, and a collection of J programs he wrote more recently to help with the actuarial management of Warranty business. But the code is entirely new, and its style is entirely different from that of the earlier projects, with the goal of making it simultaneously useful as a learning tool and as a collection of building blocks. Being new, the code certainly will not be perfect. The reader is encouraged to test the programs to try and discover imperfections, or think of enhancements, and relay ideas to the author.

The script is documented with numerous comments, each beginning with the J word `NB.` . Some of the comments explain the groups of related functions into which the script is divided. Others describe the purpose of each function and the structure of its arguments and results. Yet other comments explain some of the internal logic of the functions.

Many of the smaller functions in our script are utility functions or components from which the larger functions are constructed step by step. We have built most of the library out of such components so that the components themselves will be readily available for other uses, rather than being buried in the code of larger functions.

The script begins with a few setup instructions, which set the print precision, load the packages **plot** and **stats**, and define the "home directory" by the global variable `HDir`; this provides a convenient starting point for pathnames to the working directory or to other files with actuarial data, and should be customized by each user. The remainder of the script consists almost entirely of definitions of functions, including and interspersed with comments. The reader is encouraged to open the script on his or her screen and follow along while reading the following discussion.

3.1. Utility Functions

We start our library with some utility functions, not because they are individually that important, nor because their definitions must precede any definitions that call them (this is not the case), and certainly not because we prefer a bottom-up style of exposition. But putting some small functions first will help the reader learn J in manageable steps and understand the logic of the larger functions when we come to them. The reader is encouraged to test these functions on small vectors, matrices, etc., to see what they can do, and then to study their code to see how they do it. We pick out a very few of them to explain here.

When studying the processing flow of a tacit definition, a nice trick is to set the system to display functions in boxes (menu selection **Edit|Session Display Form|Boxed**) and then type the name of the function alone. For example:

```
   DMat
┌─┬─┬──────────┐
│]│*│┌─┬─────┬─┐│
│ │ ││[:│IMat│#│
│ │ │└─┴─────┴─┘│
└─┴─┴──────────┘
```

Function to convert a numeric vector into a diagonal matrix

It works by creating an identity matrix of side equal to the length of `y` , using the monad `IMat` in a capped fork, and then multiplying by `y`

This is a simple one that hardly needs organizing in boxes, but deeper tacit functions may require more vertical space for the nested-box display than we wish to use up on this page!

Supplying default values of arguments. We use the verb `Fill` when passing parameters to a function of more than two variables. Typically we design such functions so that either `x` or `y` or both are vectors, numeric or boxed, the *trailing items* of which may be omitted and will then be assigned default values by `Fill` . It is also possible to pass multiple related parameters as vectors within vectors, and execute `Fill` to assign default values to trailing items at each level:

```
'a b c'=.('';'';1000) Fill x
'a0 a1 a2 a3 a4'=.100 _ 0 1 2 Fill a
'b0 b1 b2 b3'=.5 20 7 _ Fill b
```

Default `c` is `1000`; defers `a` and `b`
Assigns defaults to items of `a`
Assigns defaults to items of `b`

This procedure makes it easy to organize parameters so as to avoid the need to enter many default values every time we call a function. It is usually convenient for future calls to a function if its arguments are grouped in multiple boxed vectors (rather than strung out in a single vector) and if rarely-modified parameters appear toward the end of the overall argument and of each group.

Delimited text files. Files of text delimited by commas (**.csv**, comma-separated values) are one of the built-in output formats of Excel and other commercial programs. We do not know why anyone, in the early days of computing, ever thought that the comma was a suitable delimiter for arrays converted to text files, as if it had few other uses, rarely appeared in data, and had a shape uniquely suggestive of a delimiter. Much better is to delimit files with tabs or with pipes (|). We have a family of utility functions to manage text files with arbitrary delimiters. Among other advantages, these functions make it very easy to export data and reports to, and import them from, Excel.

```
   y
```

Policy Number	Effective Date	State	Class	Premium
12345	2018 Jan 20	ME	AB	1234.56

```
   '|' ToD2 y
Policy Number|Effective Date|State|Class|Premium
12345|2018 Jan 20|ME|AB|1234.56

   ToCSV y
Policy Number,Effective Date,State,Class,Premium
12345,2018 Jan 20,ME,AB,1234.56

   y-:UnCSV ToCSV y          Does UnCSV ToCSV leave y unchanged?
1                            Yes
```

Dates. When interpreting data from outside sources, and when formatting reports for distribution, we need functions to manage dates. As a basic format for dates we use numeric yyyymmdd, as a basic format for months only we use numeric yyyymm. Converted to character strings these formats make convenient headers for columns and rows of reports.

The Julian Day Number is a mapping of dates into integers such that successive dates differ by 1. Subtracting Julian Day Numbers is a convenient way of measuring the number of days between two dates. Although the origin of Julian Day Numbers is January 1, 4713 BC, our algorithms for computing it are good only after December 31, AD 1600. Technically, these days begin at noon Universal Time; we use them to refer to conventional days beginning at midnight local time.

The conversion from Julian Day Number to calendar date, `JD` , is more complex than the conversion from date to Julian Day Number, `DJ` , so it requires a multi-line definition. Both `DJ` and `JD` have rank `0` , making them suitable for item by item conversions of vectors of dates, such as fields extracted from files.

```
   DJ 20180101 20180704 20181225
2458120 2458304 2458478
   JD 2458120 2458304 2458478
20180101 20180704 20181225
```

`DJ` and `JD` provide a simple way of checking that a numeric expression yyyymmdd represents a possible date: if `y` is such a date, then `JD DJ y` should equal `y`, otherwise not. We define the function `IsDate=: ]=[:JD[:DJ]` to test for datehood:

```
   IsDate 20180331 20180340 20181401 20180229 20000229 19000229
1 0 0 0 1 0
```

March 40, 2018, Duodecember 1, 2018, February 29. 2018, and February 29, 1900 are not permissible dates, while March 31, 2018 and February 29, 2000 are permissible. `DJ` assigns all of these Julian Day Numbers, but some properly belong to dates named differently, as is revealed by `JD` (we "walked" the result line nine spaces to the right to facilitate the comparison):

```
   JD DJ 20180331 20180340 20181401 20180229 20000229 19000229
         20180331 20180409 20190201 20180301 20000229 19000301
```

The Julian Day Number makes it easy to program functions that increment or decrement dates by a fixed number of days or that return the day of the week of any date.

```
   y=.20180101 20180704 20181225
   60 IncrD y
20180302 20180902 20190223
   DofW y
1 3 2
   NameD y
```

Monday	Wednesday	Tuesday

Some other functions in this group convert yyyymmdd dates to spelled-out formats.

```
   FmtD y
```

1 January 2018	4 July 2018	25 December 2018

Other functions convert dates in the opposite direction, from dates in various formats that may be found in fields of data from external sources, to our standard numeric format:

```
   DFld3 FmtD y
20180101 20180704 20181225
```

Calls to utility functions provided by J. Among the "general supporting functions" there are a few, such as `ToD2`, `ONames`, and `NptDiscrete`, that refer to objects with ordinary names provided by the J system, such as `LF`, `names`, and `cile` . J defines a large number of such objects, either automatically from the base library (like `LF` and `names`) or as part of add-on packages that the user may choose to load (like `cile`, in the package **stats**). These functions are not at a low enough level to be part of the J language proper, which is why they have ordinary names instead of reserved words. They are placed in the `z` locale. At run time, the search for any name starts with local names in an explicit definition, then global names in the current locale, then global names in the locales in the current locale's *path*, and eventually global names in the `z` locale.

Any object with the same name defined earlier in the search path will shadow an object we might wish to reference in the `z` locale. It is easy enough to avoid this problem with global objects by giving them distinctive names. A glance at the result of `names_z_ 0 1 2 3` (which lists the names of objects of all types in the `z` locale) suggests that we will usually be safe with names with mixed capital and lower-case letters, and always safe with such names with a capital other than the first character, at least when the only add-on packages loaded are **plot** and **stats**.

Within explicit definitions, we simply need to avoid defining local names that are the same as any names from the `z` locale that we may wish to call *from the same definition*, either directly or indirectly. For example, loading **stats** places the function `mean` in the `z` locale; it is nothing more than the archetypical fork `+/%#`. If we wish to use this function within an explicit definition (rather than just writing out the fork), then we should avoid using `mean` as a local name in that definition. Otherwise, we may use the name freely, with no effect whatever outside that definition.

We can check for indirect calls by displaying the definitions involved. For example,

```
   stddev_z_      (We check the names in _z_ in case we have overwritten them in _base_)
%:@var
   var_z_
ssdev % <:@#
   ssdev_z_
+/@:*:@dev
   dev_z_
-"_1 _ mean
   mean_z_
+/ % #
```

Here we see that, if we wish to call the `stddev` defined in the `z` locale, we should avoid overwriting not only that name but also `var`, `ssdev`, `dev`, and `mean`. In practice, name conflicts may be avoided, in explicit definitions calling objects in the `z` locale, by using the same kind of distinctive names for local variables that we mentioned above for global variables.

Any names referenced directly or indirectly in *tacit* definitions may be shadowed by names in the calling environment, which cannot be foreseen when writing the original definition. For this reason, when we define objects calling names defined by the system, or by the addon packages **stats** or **plot**, we make such definitions explicit rather than tacit. Exceptions are the names of non-printing characters `LF`, `CR`, `FF`, and `TAB`, which we allow in our tacit definitions and which should not be used for other purposes.

3.2. Miscellaneous Statistical Functions

We include a few simple statistical functions useful to actuaries in the context of building tools for larger tasks. These include several density or random-deviate functions; the functions `FD`, `FDd`, `FDi`, and `FDg`, managing frequency distributions of sample data; `NptDiscrete`, giving the centers of gravity of `x` equiprobable, mutually exclusive, and exhaustive subvectors of `y`, in ascending order; `NptDiscreteW`, a weighted version of the same thing (useful for distributions of miles driven per annum); `ACorr`, giving the lag-`x` autocorrelation of a time series or other vector `y`, and `ACorrs`, giving the first `x` autocorrelations; `Clusters`, grouping a set of weighted integers into non-overlapping groups of neighboring values subject to a minimum size and a maximum number of groups (useful in grouping policy terms for homogeneity); `KDens`, calculating kernel density functions, `PlotDens`, comparing a sample density represented by a histogram with one or more kernel density estimates; and `ExpTrend`, fitting an exponential trend line or lines to a time series of ratios of a vector of losses (or similar quantities) to a vector of exposures (or similar weights).

This last function is called by some higher-level functions, discussed below, that estimate trends in severity, frequency, loss ratios, or pure premiums. It returns parameter estimates and related statistics and plots the results to screen or file. It can derive a single trend curve, with trend uniform throughout the experience period, or can include one or more change points with different trends in each interval. It does not detect the need for change points, nor quantify that need with a confidence level, but it can search for an optimal set of change points by minimizing the sum of squared deviations from the fitted curves. The user may identify the need for change points from knowledge of changes in contracts or claims administration, or by running the program with a single trend and inspecting the residuals.

3.3. Management of Source Data

Nearly every reserving or ratemaking project relies on experience data, which in its raw form usually consists of one file with policy information and one with claims information, with at least

one field in the claims file serving as a link to the policy file. Nearly always, the files in question are, or may easily be translated into, delimited text files. The details of these files will, however, differ, from one source to another, in the number and order of fields, the location and contents of column headers, the format of dates, and so forth. And the files may be extremely large, burdened with fields necessary for administration but of little or no use in reserving or ratemaking.

We have found it convenient to transform the data for each project into a common intermediate file format and code our calculation functions to expect data in that format. The intermediate format captures the fields commonly needed for ratemaking and reserving and gives them predefined names, while allowing the user to capture additional fields that may be useful for a specific case and give them customized names. This is equivalent to selecting and renaming the useful fields in the original table, while discarding the unnecessary fields, but *we do not leave the results in tabular format*. Instead we create a single file for each policy field, and a single file for each claims field, and a single file identifying by index the policy associated with each claim. These files have the extension **.csv** (comma-separated values), but, as they are column vectors, the only delimiters they actually employ are linefeeds. The files contain data only; the filenames serve as headers. All of the policy files and all of the claims files have a common length and a common order, because they were originally fields in a common table, or were derived element by element from such fields.

For calculations and reports, we load each required file separately, converting it either to a numeric vector or to a vector of boxed character strings, and we give the resulting global variable the same name as the file, but without the **.csv** extension. The variable and file names are chosen so as to be unlikely to be assigned to any other variable or function, either globally or locally within a definition. In this library, the names of policy variables begin with lower-case `p`, followed by a string suggestive of the meaning of the variable, with its first letter capitalized. The names of claims variables are similar, but begin with a lower-case `c`.

The variables created are global so that they may be read by any function without needing to be passed as parameters. The files for any given project are expected to reside in a common directory (folder); the variable `WDir` (working directory) points to this. The function `ImportVs` brings into the workspace, from the working directory, the variables named in a list `y`, *but only those that are not already present*. The function `SetWDir` assigns a new value to `WDir` and erases all `p` and `c` variables in the workspace; this ensures that all such variables subsequently used will be from the same source, and for this reason `SetWDir` should be used instead of defining `WDir` manually.

Occasionally there is need for a variable derived from the `p` or `c` variables but not of the same shape as either: for example, lookup tables for UPR formulas, discussed in a later section. In order that such variables may be cleared from the workspace by `SetWDir`, we name them analogously to

the `p` and `c` variables, but with initial letter `s` (for "special"). `SetWDir` will erase any such variables found, while functions, such as `TrimData`, that detect the `p` and `c` variables and rely on their having known shapes, will not be affected.

Policy variables include *identifying information*, e.g. policy numbers; *classification information*, e.g. state; *contract information*, e.g. term; *date information*, e.g. issue date; and *amount information*, e.g. premium. Claims variables include a vector of *indices into the policy file*, often derived from policy numbers included in the claims file; *date information*, e.g. accident date; and *amount information*, e.g. paid amounts. For lines with case reserves valued multiple times, or with multiple payments, the claims files may have one item for each transaction, rather than one for each claim; in this case there needs to be identifying information for the claims, such as claim number, or a reasonable number of additional fields to accommodate second and subsequent valuation or payment dates and amounts.

With our intermediate file format at hand, we only need to write one file-preparation function for each project; from that point on we may ignore the original source files. Some calculations require only a few variables; these may be brought into memory without wasting space on unnecessary fields. We have used a system similar to this for Warranty business for years, handling with no difficulty, in an ordinary PC, files with several millions of contracts or claims. Even larger policy data may be summarized over one or more classification and date fields, normally by including a "counts" field in the policy file.

The function `StdNames` lists those `p` and/or `c` names that are standardized for reference by other definitions in this library. For example, `pT` refers to the nominal term in months of a Warranty contract, which is expected by our functions to have this name. Files specific to a single project may be given arbitrary names, referenced by functions written for that project, but should normally follow the `p` or `c` naming convention so they will be cleared from the workspace by `SetWDir`.

These standard variables have been chosen for their usefulness in Warranty and related lines, where there may be millions of policy and claim records; where the term of each contract, and of the underlying factory warranties, are essential; where claims usually involve single payments; where report and settlement lags are much shorter than lags from issue to accident, and loss reserves of all kinds much smaller than the Unearned Premium Reserve; and where the very terminology reflects the peculiarities of the coverage, with for example "failure date" or "breakdown date" commonly used in lieu of "accident date". But the user may easily adapt the intermediate file format to other lines of insurance, with the main difference being the need for transaction files to record the dates and amounts of changes in case reserves.

Many of the standard variables are included in almost any set of policy and claims files. In particular, these include the nominal term and manufacturers' warranties, in both months and miles where appropriate. Others do not usually appear in the source data proper but are created after the source data has been converted to intermediate files. Examples of these are `pAT`, the actual term from issue to expiration, which sometimes differs from the nominal term, and `pRMW1`, the remaining MW for power train, which, for a used vehicle, usually differs from the original MW. For actuarial purposes the actual terms and remaining MW's are almost always preferable to their nominal counterparts, except occasionally when filtering or grouping data. Creating these variables once as the data is being assembled avoids the need to re-create them for every new analysis.

Our main function for creating intermediate files is `SplitFields`, which extracts desired fields from a delimited text file and creates separate single-field files. Its arguments specify, among other things, the headers of the requested columns in the source file, the names of the single-variable files to be created from them, and the function, if any, to be used to process each field, for example to translate dates from their source format to yyyymmdd. The typical approach to bringing in data for a new project is to define a small cover function which constructs the arguments for, and then calls, `SplitFields` . These cover functions may be saved for later use with similar projects. We include a template for such a cover function, `SetUpData`, as an example.

The function `CompleteData` derives variables such as `pAT`, `pRMW1`, and `pCt`, which may not be actual fields in the source data, and saves them to file.

The function `CleanseData` examines the policy and claims variables for consistency in dates and reasonableness in amounts, and the claims variables for successful matches to policy records, and creates the Boolean variables and files `pGood` and `cGood` flagging the policy and claims records that "pass". This avoids the need for such data testing with every report that is run.

3.4. Credibility Theory

We have included a set of credibility functions in our library because of the usefulness of credibility theory in ratemaking. The main function is `JHC`, Jewell's Hierarchical Credibility, which may also be used for single-level credibility using the Bühlmann-Straub and related models. This function itself does the numeric calculations; it is surrounded by higher-level functions to produce formatted reports and lower-level functions to simplify the management of data.

To organize data for hierarchical credibility, the usual "rectangular" multidimensional arrays of J are not the best choice, as we are dealing with nested rather than crossed data. A tree structure is better, each node of the tree representing one level of the hierarchy. The nodes may have different numbers of branches (different classes having different numbers of subclasses), but we assume that

all paths through the tree from root to leaves go through the same number of nodes (every policy has a value at every level). A good example, from Auto Warranty, has, for levels, manufacturer, make, and model. Every car has each of these attributes, if necessary by repeating the name of the higher level when there is only one make for a manufacturer, or one model of a make. The lowest level in hierarchical credibility usually represents time periods. The complete tree in our example might therefore have nodes for manufacturer, make, model, and year.

We have found nested vectors of boxes to be a good representation of such a tree. At the top is one box for each value at the top level. Within each of these boxes is one box for each nested value at the next level down, and so forth. Eventually we come to the lowest level, with a numeric matrix of weights and ratios by time period. This is all the *numeric* and *structural* information we need for hierarchical credibility calculations. When preparing formatted reports, we also need *interpretive* information: a name for each level (e.g. manufacturer, make, model) and a name for each *value* found at each level of each branch of the hierarchy (e.g. Toyota, GM, Volkswagen, …; Toyota and Lexus within Toyota; Avalon, Camry, Prius, … within Toyota/Toyota). The names of levels may be taken from the names of the classification variables. The values are usually as found in the data.

`JHC` implements hierarchical credibility as described by Belhadi, Goulet, and Ouellet [2] and by Dutang, Goulet, Milhaud, and Pigeon [5], including the iterative pseudo-estimator and the Bühlmann-Gisler and Ohlsson approximate estimators of the variance parameters. It calls itself recursively in such a way that each call needs to estimate only the mean "within" variance (σ^2 in the literature) and the "between" variance (*a* in the literature). There is no need to calculate the next-level between variance (*b* in the literature) separately, as it is simply the *a* parameter in the next recursive call to `JHC`, while *a* becomes the estimate of the σ^2 parameter at that level. The recursion proceeds to the top of the hierarchy and then cascades back with the results, which include the predicted ratio at the next level above any given level; this becomes the "class mean" to be given weight `1-Z` at the given level.

The function `MakeHierarchy` constructs a nested hierarchy of classification variables and time periods, suitable for a right argument to `JHC`, given matrices of the weight, loss, and date variables, and of any classification variables. For each combination of classifications, it includes all time periods, even those with zero weight. This is not strictly necessary; the algorithm for estimating σ^2 eliminates years with zero weight when counting degrees of freedom, and the zero weights themselves eliminate those years from the sum of squared deviations, so it would produce exactly the same results if the years with zero weight were omitted from the nested boxes. We include all time periods to facilitate models involving trending.

The function `JHCRpt` converts the result of `JHC`, together with information on level and value names, into a formatted report with multiple tables. `JHCRpt1` runs `JHCRpt` and flattens the result into a single table. The function `SevCred` uses source data in our standard intermediate file format to create, through `MakeHierarchy`, arguments to `JHCRpt1` to create a credibility analysis of severity. The function `LRCred` performs a similar analysis of loss ratios, loss costs, or frequencies, depending on the choice of loss and exposure measures, developing earned exposure by use of a UPR formula.

Example. Hachemeister, in a 1975 paper on credibility, included for illustration a small data set of claim counts and severities; this data set has been picked up by subsequent authors. Originally it had a single level, "state" (and included just five states); Dutang, et al [5] separated the states into two "cohorts" so as to illustrate hierarchical credibility as implemented in R. We have included this data set in the global variables `HachX` (severities) and `HachW` (weights, or claim counts), and have converted them into a right argument to `JHC` for a single-level analysis under the name `Hach`, and for a two-level analysis under the name `Hach2`. If we execute

```
Rpt=.('';'';<'Cohort';'State') JHCRpt1 Hach2,
```

convert `Rpt` to a **.csv** file and pass it through Excel to format it nicely, we get the results shown in the following table, which replicate those of Dutang, et al exactly.

Cohort	State	Weight	Ratio	Z	Class Mean	Cred Ratio
0	0	100155	2061	0.8874	1949	2048
0	1	13735	1806	0.5195	1949	1875
1	0	19895	1511	0.6103	1543	1524
1	1	4152	1353	0.2463	1543	1497
1	2	36110	1600	0.7398	1543	1585
Within variance	139120026					
Between variance	10952					
Cohort	**Weight**	**Ratio**	**Z**	**Class Mean**	**Cred Ratio**	
0	1.407	1967	0.9196	1746	1949	
1	1.5964	1528	0.9284	1746	1543	
Within variance	10952					
Between variance	88981					

3.5. Loss Development

The development of loss triangles is at the core of casualty loss reserving both for reserves *per se* and for ultimate loss estimates in ratemaking. We provide functions to develop triangles using the Bornhuetter-Ferguson family of estimators, the Partial Loss Ratios family, and several recently-published estimators that attempt, among other things, to reconcile paid and incurred loss projections for the same portfolio.

In the Bornhuetter-Ferguson (BF) family we include all estimators using the traditional chain-ladder (CL) method for lag factors and the Cape Cod, or Stanard-Bühlmann, estimator for expected loss ratios (ELR's). By using decay factors as suggested by Gluck [6], the BF family includes the pure CL estimator at one end (Gluck factor 0) and the pure Cape Cod at the other (Gluck factor 1). We also include the pure CL estimator with nonparametric estimates of the standard errors of the reserves, from Mack [8].

The Partial Loss Ratios (PLR) family (called Complementary Loss Ratios by Bühlmann and Additive by Stanard) is analogous to the BF family except that lag factors are estimated via an additive model, with column effects only, applied to the triangle of partial loss ratios by lag.

The recent models include Yamashiro's Recursive Credibility model [15], Quarg and Mack's Munich Chain Ladder model [11], Merz and Wüthrich's Paid-Incurred model [9], the Double Chain Ladder and related models of Agbeko et al [1], and Müller's Affine Loss Development model [10]. Obviously there are many more estimators we might have included (and might yet include), such as the overdispersed Poisson models of Verrall and others, but we cannot include everything. This may be a library of actuarial models, but it is not the *Bibliothèque nationale* of such models.

By "triangle" we mean the usual matrix of losses or other quantities by accident or issue period versus development period or development lag. Such triangles may contain many different quantities: exposures, reported claim counts, paid claim counts, incurred losses, paid losses, loss ratios, etc. A triangle may be lagged or not lagged; may be cumulative or incremental; its first axis may represent issue, accident, or report period; its second axis may represent accident, report, or payment period; its cells may be of any size, such as one, three, or twelve months; its time periods on the first dimension will have a lower and an upper bound, and, on the second dimension, an upper bound; it may contain just known values or known and projected values. To capture all these variations we define a "triangle" object to be a vector of eleven boxes, the first of which contains the triangle proper and the next ten of which contain interpretive values or codes. This structure is explained in detail in the script.

We define a group of functions that create triangle objects from amount and date variables (`BuildTri`), complete them with default values for trailing items (`FillTri`), convert them from one representation to another (`LagTri`, `UnlagTri`, `CumTri`, `DiffTri`), increase their cell size (`AggrTri`), extract portions of them (`KnownTri`, `BandTri`, `FutureTri`), modify their values (`TrendTri`, `RoundTri`), and format them for display (`DispTri`).

Next we define the core calculation function `LDFs`, using chain-ladder to estimate parameters of the loss process in the lag direction: loss development factors, ultimate loss development factors, completion factors, and lag factors. The similar function `PLRs` estimates completion and lag factors using partial loss ratios. Each of these functions offers certain options useful in the context of calculating the Unearned Premium Reserve (UPR) for Warranty and similar long-duration lines: they allow for exposure that varies by development as well as accident period; they correct for unreported losses when developing an issue-period by accident-lag triangle; and they allow for residual loss development relative to an assumed matrix of *a-priori* lag factors. The last option improves the response of the estimator to lack of homogeneity over time and simplifies the weighting of experience factors with reference factors and the extension of experience factors with tails; see Vaughan [14]. Both LDFs and PLRs will produce a step-by-step report if requested.

Completing a loss triangle also requires row parameters. The function `CapeCod` estimates ELR's, given a set of lag factors, with the Gluck adjustment permitting the full range from pure CL to pure Cape Cod.

The final completion of a triangle, or of a set of related triangles, is performed by a set of functions that we call loss projection programs; these have a standardized input and output structure so that they may be called by name from within higher-level functions, without the latter's knowing their details beyond the standard inputs and outputs. At present the available loss projection programs are `PaidBF`, `IncdBF`, `Mack`, `MCL`, `MerzWuethrich`, `Yamashiro`, `DCL`, and `AffineLD`. These may be modified by left arguments (attached to the name with the bond conjunction `&` when passed through a higher-level function), so each is actually a family of estimators. For example, supplying different Gluck factors allows `PaidBF` and `IncdBF` to range between pure Chain Ladder at one extreme and pure Bornhuetter-Ferguson at the other. The library includes the higher-level function `LDReport`, which runs a set of loss projection programs on the same data, assembling the results into a table which compares reserves, ultimate losses, and (where available) standard errors and standard errors relative to reserves.

The result of a loss projection program includes the completed *paid* triangle, a summary matrix of losses paid to date, reserves, ultimate losses, standard errors, and relative standard errors, a description of the estimator, and certain intermediate results both numeric and formatted. Some of

the estimators that use incurred data do not automatically generate a completed paid loss triangle, with every cell populated, but may satisfy themselves with ultimate losses (and therefore reserves) by accident period. In this case the summary matrix contains the direct results of the estimator, while the completed paid triangle may be derived from a separate projection of paid losses, with reserves rescaled to the level of the incurred projection.

Toward the end of the library script we have defined several of the loss triangles used as illustrations in the source papers; these are convenient for validating the loss projection programs and for experimentation. For example, both Quarg and Mack [11] and Yamashiro [15] use a seven-year set of paid and incurred data from a fire portfolio; we have named these triangles `MCLPaid` and `MCLIncd`. If, setting up MCL to use Quarg and Mack's judgment sigmas, we run

```
A=:('(0.1;0.1)&MCL';'Yamashiro') LDReport MCLPaid;<MCLIncd
```

we obtain the following results (formatted by passing through Excel):

		A		B		C				D			
Period	ITD Paid	Reserve	Ultimate	Reserve	Ultimate	Reserve	Ultimate	StdErr	StdErr/Res	Reserve	Ultimate	StdErr	StdErr/Res
1980	2131	0	2131	43	2174	0	2131	0	0	43	2174	0	0
1981	2348	35	2383	96	2444	37	2385	17	0.007	87	2435	20	0.0083
1982	4494	103	4597	135	4629	116	4610	52	0.0114	207	4701	56	0.0119
1983	5850	269	6119	326	6176	276	6126	79	0.0129	400	6250	82	0.0131
1984	4648	289	4937	302	4950	328	4976	75	0.0151	427	5075	78	0.0153
1985	4010	646	4656	655	4665	610	4620	156	0.0337	704	4714	159	0.0337
1986	2044	5504	7548	5606	7650	5136	7180	466	0.0648	5281	7325	475	0.0648
Total	25525	6846	32371	7162	32687	6504	32029	-	-	7149	32674	-	-

A: Munich Chain Ladder: paid indications
B: Munich Chain Ladder: incurred indications
C: Yamashiro's Recursive Credibility Model: paid submodel
D: Yamashiro's Recursive Credibility Model: incurred submodel

These results agree exactly with the illustrations in Quarg and Mack and in Yamashiro, except for a difference of 1 in the units place of the 1986 ultimate losses for "A", apparently due to rounding.

Both Mack and Müller use the so-called Taylor-Ashe reinsurance triangle as an example (it started in a 1983 paper by Gregory Taylor and F.A. Ashe and has a long history of such use), and, if we run

```
A=:('Mack';'AffineLD') LDReport ,<TaylorAshe
```

we obtain the following table, which reproduces their results (except that Müller rounded the triangle to the nearest £1,000):

		A				B	
Period	**ITD Paid**	**Reserve**	**Ultimate**	**StdErr**	**StdErr/Res**	**Reserve**	**Ultimate**
1972	3901463	0	3901463	0	0	0	3901463
1973	5339085	94634	5433719	75535	0.7982	94634	5433719
1974	4909315	469511	5378826	121699	0.2592	518869	5428184
1975	4588268	709638	5297906	133549	0.1882	777210	5365478
1976	3873311	984889	4858200	261406	0.2654	979089	4852400
1977	3691712	1419459	5111171	411010	0.2896	1418592	5110304
1978	3483130	2177641	5660771	558317	0.2564	1941919	5425049
1979	2864498	3920301	6784799	875328	0.2233	3664065	6528563
1980	1363294	4278972	5642266	971258	0.2270	4225048	5588342
1981	344014	4625811	4969825	1363155	0.2947	5026082	5370096
Total	34358090	18680856	53038946	2447095	0.1310	18645509	53003599
A: Chain-ladder with Thomas Mack's non-parametric estimates of standard errors							
B: Affine Loss Development: paid affine model							

Merz and Wüthrich provide an example with a pair of triangles, paid and incurred. The following table, generated by running

```
A=:(,<'MerzWuethrich') LDReport MWPaid;<MWIncd
```

reproduces their results, *but only approximately!* Some of our reserves differ from theirs by small amounts, which might be explained by rounding, but the paid, incurred and combined standard errors differ by 0.3%, 8.3% and 7.1%, respectively. This must have been fated to be a challenge to our readers help us reconcile the difference, modifying `MerzWuethrich` if necessary.

		A				B				C			
Period	**ITD Paid**	**Reserve**	**Ultimate**	**StdErr**	**StdErr/Res**	**Reserve**	**Ultimate**	**StdErr**	**StdErr/Res**	**Reserve**	**Ultimate**	**StdErr**	**StdErr/Res**
2008	3921258	0	3921258	0	0	0	3921258	0	0	0	3921258	0	0
2009	2567056	115470	2682526	186478	0.0695	337993	2905049	65143	0.0224	337799	2904855	66717	0.0230
2010	3182511	428272	3610783	298173	0.0826	31525	3214036	53427	0.0166	31685	3214196	53733	0.0167
2011	3003425	642664	3646089	297678	0.0816	331522	3334947	22714	0.0068	331886	3335311	22666	0.0068
2012	2150351	729344	2879695	253094	0.0879	1018920	3169271	15799	0.0050	1018303	3168654	15770	0.0050
2013	2385339	1284545	3669884	335244	0.0913	1102551	3487890	31113	0.0089	1104788	3490127	30986	0.0089
2014	1487577	1183781	2671358	252044	0.0944	1869215	3356792	50412	0.0150	1842603	3330180	49405	0.0148
2015	1484405	1692632	3177037	357001	0.1124	1989783	3474188	149870	0.0431	1953351	3437756	140259	0.0408
2016	1376124	2407429	3783553	505950	0.1337	1464946	2841070	172502	0.0607	1601572	2977696	170150	0.0571
2017	841930	2027245	2869175	415212	0.1447	2546822	3388752	277802	0.0820	2401783	3243713	246994	0.0761
Total	22399976	10511381	32911357	1554720	0.0472	10693277	33093253	386350	0.0117	10623769	33023745	361691	0.0110
A: Merz and WÃ¼thrich's PIC model: paid only													
B: Merz and WÃ¼thrich's PIC model: incurred only													
C: Merz and WÃ¼thrich's PIC Model: paid and incurred													

Merz and Wüthrich's model implemented here is their original 2010 version, not the 2013 versions by Happ and Wüthrich modeling dependence within the triangles, nor the 2016 version by Merz and Wüthrich including tail factors.

Agbeko et al require paid and incurred losses and reported claim counts for their DCL model, and provide two examples, one of bodily injury data and one of property damage data. The PD set

is a bit peculiar, with cumulative paid losses from the first year becoming negative at lag five years. Here we apply the DCL methods to their BI data set, by running

```
A=:(,<'DCL') LDReport DCLBIPaid;DCLBIIncd;<DCLBIClaims
```

to obtain the following table:

		A		B		C	
Period	**ITD Paid**	**Reserve**	**Ultimate**	**Reserve**	**Ultimate**	**Reserve**	**Ultimate**
1998	2781	0	2781	0	2781	0	2781
1999	5446	0	5446	0	5446	-1	5445
2000	7964	-1	7963	-1	7963	-2	7962
2001	10597	0	10597	0	10597	15	10612
2002	16452	0	16452	0	16452	52	16504
2003	20090	50	20140	50	20140	37	20127
2004	29204	90	29294	90	29294	74	29278
2005	35332	181	35513	181	35513	138	35470
2006	37588	267	37855	266	37854	130	37718
2007	36801	322	37123	321	37122	208	37009
2008	38577	382	38959	380	38957	255	38832
2009	35343	445	35788	438	35781	-114	35229
2010	30425	502	30927	507	30932	882	31307
2011	37189	1186	38375	1206	38395	1857	39046
2012	39034	2391	41425	2445	41479	3452	42486
2013	39328	6463	45791	5998	45326	2423	41751
2014	27054	10826	37880	9548	36602	5808	32862
2015	26854	23908	50762	19131	45985	13857	40711
2016	16823	37103	53926	28329	45152	24996	41819
2017	4614	86338	90952	41617	46231	39227	43841
Total	497496	170454	667950	110506	608002	93292	590788
A: Double Chain Ladder (DCL)							
B: Bornhuetter-Ferguson Double Chain Ladder (BDCL)							
C: Incurred Double Chain Ladder (IDCL)							

This reproduces the results of Agbeko, et al, but once again only approximately; the totals differ by 0.1%, 0.3%, and 0.2%, respectively, from those published in the paper, likely the effect of internal rounding in Agbeko's calculations.

3.6. Unearned Premium Reserves

The Unearned Premium Reserve (UPR) for long-duration contracts is regulated to prevent the premature taking of single premiums into earnings. Among other things, the regulations require that the aggregate UPR be no less than the gross written premium multiplied by the expected unincurred fraction, at the accounting date, of ultimate losses and expenses. This is a felicitous rule, as it leads to ITD loss ratios that are likely to be good predictors of ultimate loss ratios, and most insurers attempt to satisfy it on a contract-by-contract basis as well as in aggregate. Typically, they calculate the UPR for individual contracts by formula, and validate their formulas from aggregate experience.

Our focus here is, first, on estimating, from experience, UPR "curves" (monthly Unearned Premium Factors, or UPF's, from issue to expiration); second, on developing formulas for UPF's dependent on the provisions of each contract and the characteristics of the insured; and, third, on developing metrics to compare the adequacy and accuracy of different UPR formulas relative to experience curves or to each other. For concreteness we take this in the context of warranties, or extended service contracts, which are typically written for a long duration with a single premium.

Terms. Most commonly an extended service contract has a single term for all types of loss, while the manufacturer's warranty (MW), which pays first, may distinguish between parts and labor, or between power train and other failures. Terms for most consumer products are given in months, but terms for vehicles in months and miles, whichever expires first. These *nominal terms* are kept few in number for simplicity in marketing. However, the nominal term may begin at several different possible times, such as when the product is purchased, when the contract is purchased, or when the manufacturer's warranty ends. The *actual term*, from issue to expiration, which is of primary interest to the insurer, therefore depends on both the length and the starting date of the nominal term.

Dates. Several dates are necessary to make each contract precise: the *in-service date*, when the manufacturer's warranty begins to run; the *issue date*, when the contract is first in force and the UPR is first required; the *effective date*, when the nominal term begins to run, and, for a cancelled contract, the *cancellation date*, when a refund may be payable and when any remaining UPR is released Additional dates may appear in the data, such as *MW end date* and *expiration date*; they give some redundancy useful in checking data integrity. Similarly, claims typically require *breakdown* (*failure*, or *accident*) *date*, *report date*, and *payment date*, though, for reasons of data reliability, the payment date is often treated as the report date, and paid rather than case incurred losses used in the analysis.

Programs. The portion of the script **act.ijs** dealing with the UPR contains several utility functions applicable in this context alone (`EPFtoUPF`, `UPFtoEPF`, `AdequacyRatios`, `PlotUPFs`, etc.), some utility functions applicable in this context and for other reports based on data stored in standard format (`DateRange`, `MakeFilter`, `ClusterTerms`, etc.), several high-level

calculation functions for estimating UPR factors or earned premium factors from experience (`EPFs`, `UPFs`, etc.), similar functions for calculating such factors by formula (`FEPFs`, `FUPFs`, etc.), and some functions coding the formulas themselves.

To improve performance with large data sets, some UPR formulas are pre-calculated and converted to lookup tables and index vectors. This process is managed by `DoUPRFmla`, which creates the lookup tables and index vectors when first called and references them afterward. The formulas coded in this manner include `ProRata` (earnings uniform, UPR linear), `PRafterMW`, (pro-rata after MW), `R78` (Rule of 78), `RR78` (reverse Rule of 78), and `ApproxATF`.

The formula `ATF` (All-Terms Factors) is coded directly rather than using a lookup table, because a complete lookup table with term and two MW's in months and miles, with initial odometer reading, and with all the necessary lags, might involve many more cells than the data itself. The All-Terms Factors method may be based on any exposure model (e.g. one of our other UPR formulas), but it is most commonly based on an exposure model for Auto Warranty due to Kerper and Bowron [7]. It adjusts this basis by residual earnings factors, one for each term and lag, accounting for some of the variability not explained by the exposure model. The residual factors are calculated for *each term* within a stipulated range based on weighted triangles from *all terms* found in the data, hence the name. The formula `KBF` is a special case of `ATF` using the Kerper-Bowron exposure model with no residual adjustment.

Procedure. A typical test of a company's UPFs will first involve splitting the data into reasonably homogeneous groups of terms, then comparing indicated average UPF's in each group by experience with indicated average UPF's by whatever formula the company is using or contemplating.

Experience indications for clustered terms are given by the function `CUPFs`, which, for each cluster, calls the function `UPFs`, which calls the function `EPFs`, which estimates incremental earned premium factors by lag., factoring out the effect of cancellations.

Formula indications for clustered terms are given by the function `CFUPFs`, which, for each cluster, calls the function `FUPFs`, which calls the function `FEPFs`, which calls the function <formula>; `FEPFs` ultimately calculates incremental earned premium factors by lag, assuming no cancellations.

The user need not be concerned with all these internal function calls; he or she simply invokes `CUPFs` or `CFUPFs` on the command line with appropriate parameters.

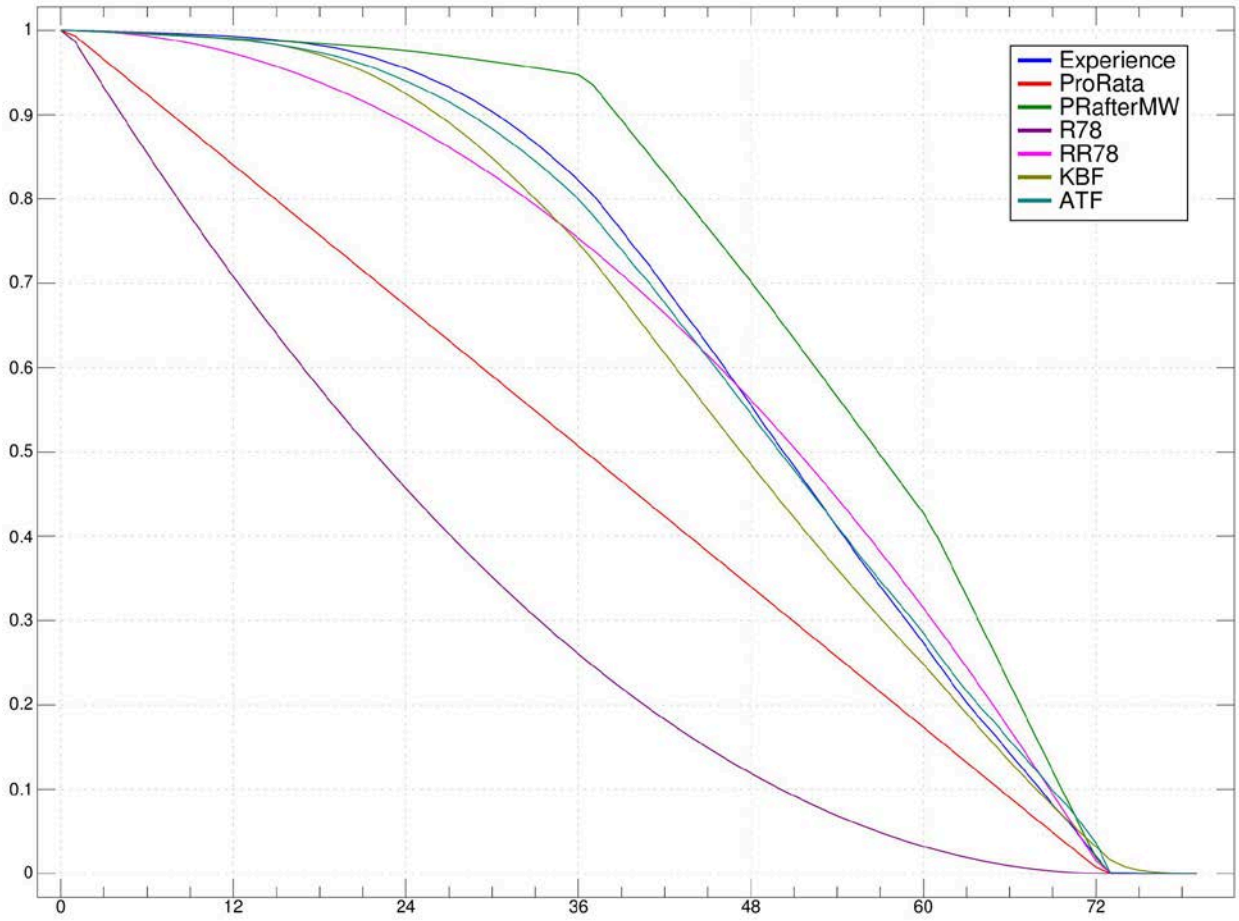
The comparisons are made with `PlotUPFs`, `AdequacyRatios`, and `LRRatios` – the first to give a clear visual indication whether a formula is more or less conservative than experience, and

whether it has approximately the right shape, the second to quantify the relative adequacy of two or more sets of UPF's, and the third to measure the effect of differences in the shape of the curves on ITD loss ratios of a book of business as it matures.

Example. For an example we use a subset of some Auto Warranty data, the provenance of which shall remain anonymous to protect the innocent. The two plots and the table below illustrate the result of running the following programs, with default arguments except for the use of a decay factor of 0.8 instead of 0.95, and a credibility constant of 500 contract months instead of 0, in `RATFs` (to minimize the influence of very thin data in the later lags of the terms above about 84 months):

1. `KBFtotals`, to produce and save a vector `pKBF` of incremental earned contract months from inception to expiration, one for each policy record, by the Kerper-Bowron exposure model. This serves as a reference measure of exposure, or an artificial premium, independent of actual changes that may have taken place in rate adequacy.
2. `RATFs`, to produce and save a table `sRATFs` of residual ATF earnings factors, one for each term and lag. This corrects the ATF model for the effect of variables not accounted for by `pKBF`.
3. `ClusterTerms`, to subdivide the data into a reasonable number of reasonably homogeneous term groups. We choose one of the resulting clusters, actual terms from 67 to 78 months, for illustration. In a real evaluation of a company's UPR factors, all clusters would be tested and `ClusterTerms` would normally run inside `CEPFs` or `CFEPFs`.
4. `UPFs`, restricted to terms between 67 and 78 months, to estimate UPF's for the chosen cluster from experience, in the absence of cancellations.
5. `FUPFs`, successively for the formulas `ProRata`, `PRafterMW`, `R78`, `RR78`, `KBF`, and `ATF`, to estimate UPF's by each of these formulas in the absence of cancellations.
6. `PlotUPFs`, to plot together the results of (4) and (5). This permits a good visual comparison of formulas versus experience (and of formula versus formula).
7. `AdequacyRatios`, to compare the relative adequacy of the various curves, pair by pair.
8. `LRRatios`, to compare the effect of the use of various UPR formulas on the UPR and the ITD loss ratios for a block of new business as it matures.

No one would consider curves such as `ProRata` or `R78` reasonable candidates for a UPR formula for Auto Warranty. But this is just an illustration! We wanted to show their shapes.

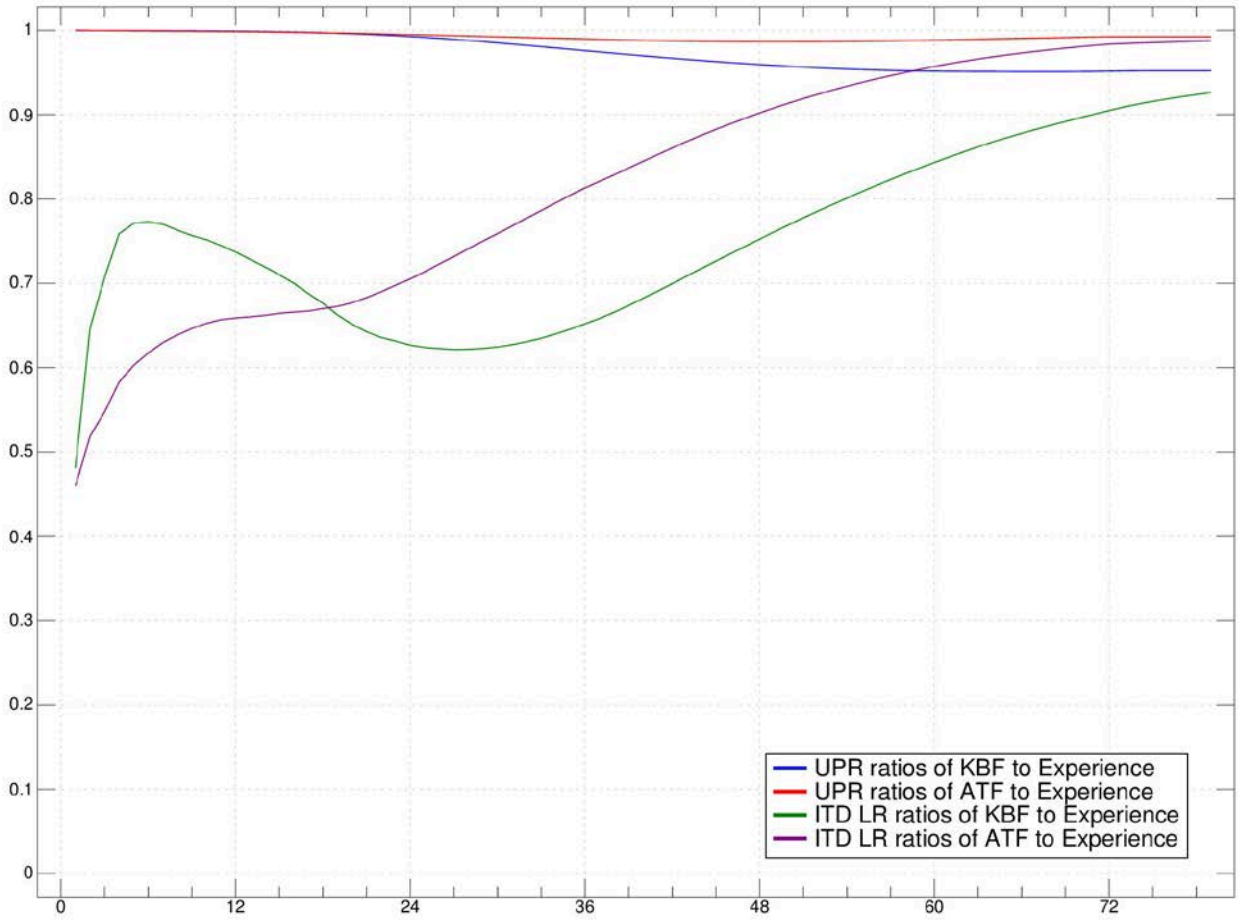


Here the KBF and ATF curves fit the experience better than the other curves. The KBF fit would probably be improved with trend assumptions greater than the default 1.02 per 10000 miles and 1.04 per annum. The residual adjustment in the ATF curve picks this up (along with any other factors not accounted for) and comes quite close to experience. The PRafterMW curve is too conservative because of not recognizing the expiration of MW's in miles. The remaining curves – ProRata (linear) and R78 and RR78 (quadratic) have no theoretical reason to apply to Auto Warranty and are shown for illustration only.

The following table of adequacy ratios shows that KBF is about 4.8% light, and ATF about 0.8% light, relative to experience, on business written at a constant rate long enough to reach a steady state. This is a convenient metric for comparing entire UPR curves, though it must be supplemented with inspection of the shape of the curves. Here the RR78 curve appears to outperform KBF, but it is very light (earns too fast) in the early lags, and somewhat heavy in the later lags; the errors offset each other and make the overall adequacy ratio close to 1.

	Experience	ProRata	PRafterMW	R78	RR78	KBF	ATF
Experience	1	1.356	0.904	2.007	1.024	1.051	1.008
ProRata	0.737	1	0.667	1.480	0.755	0.775	0.743
PRafterMW	1.106	1.500	1	2.219	1.133	1.162	1.114
R78	0.498	0.676	0.451	1	0.510	0.524	0.502
RR78	0.976	1.324	0.883	1.959	1	1.026	0.984
KBF	0.952	1.291	0.861	1.910	0.975	1	0.959
ATF	0.992	1.346	0.897	1.992	1.016	1.043	1

The following loss-ratio-ratios plot for this example illustrates the leverage that even slight differences in UPR curves may have on early ITD loss ratios. This example shows the risk of overestimating the profitability of a new book of business from its early loss ratios.



The potential for underestimation is dramatic. We must bear in mind that "early" in terms of loss emergence is different from "early" in terms of time. For example, after the block of business has been written for 18 months, the loss ratios of KBF and ATF are both only about 67% of the loss ratios by experience. We might think that, at 18 months, we were 25% of the way to ultimate,

or at least 12.5% of the way considering that the average maturity of the inforce business is 9 months. But at the average maturity of 9 months the business written to date is only about 0.5% earned. Extrapolating from ITD loss ratios would be a bit of an adventure if it really involved a ratio of only 4:1; it is pure gambling with a ratio of 200:1.

3.7. Experience Reports for Long-duration Contracts

In addition to functions such as `PlotUPFs`, `AdequacyRatios`, and `LRRatios`, which display particular statistics, we illustrate the possibility of more comprehensive reports summarizing and projecting the experience of long-duration contracts in detail.

An example is the function `ProjMonths`. This name, meaning "project months", is a bit misleading, since the function includes historical as well as projected months and since it displays, not months themselves (hard to do) but accounting items *by* month. But we needed a name.

The basic idea underlying this function is that it performs separate projections of *actual transactions* (e.g. paid losses) and of *accounting entries* (e.g. the carried UPR). The actual transactions are the result of the *loss process*, the parameters of which must be estimated. Examples of these parameters are the lag factors describing the emergence of incurred, reported, and paid losses. The accounting entries, on the other hand, depend on rules, which amount to *assumed* parameters. Of these, the UPR strings or formulas usually have the greatest impact on an underwriter's statements.

For this reason `ProjMonths` requires two separate specifications of the UPR: one to describe the actual loss process, and one to describe the UPR carried on the books. Normally the former will be based on experience (and will call our function `CUPFs`) while the latter will be based on formula (and will call `CFUPFs`), although there is nothing to prevent the user from deciding that a particular formula describes experience even better than analysis by loss development, especially if the volume of experience is small. The program is more relaxed in its treatment of loss reserves, which it derives from experience both for actual transactions and for accounting entries; this reflects the judgment that most underwriters calculate loss reserves from triangles at each valuation or, at least, update their development factors frequently.

`ProjMonths` allows the user to specify the historical data to be used, by date ranges (earliest issue date, latest issue date, latest transaction date) and by filter (to restrict the analysis to a particular collection of contracts). It subdivides the analysis and results into term groups, which the user may specify explicitly or may allow the program to determine by calling `ClusterTerms`. It further subdivides the results (but not the analysis) into issue-month cohorts, for example years ending June 30, which the user may specify. Experience has shown that underwriters and obligors may wish to

track experience by issue cohorts, typically one year in length, but that it is necessary to combine the cohorts when estimating the parameters of the loss process.

For each combination of term group (or all term groups together) and cohort (or all issue months together), `ProjMonths` returns a table whose rows are financial statement items and whose columns are calendar months. The financial statement items modify conventional accounting somewhat. For example, we distinguish *pure* written premium, for actual new contracts written, from *statement* written premium, net of refunds. Statement written premium is most inconvenient for actuarial analysis as it confounds the experience of the issue months generating the new policies with that of earlier issue months from which some of the refunds may have been derived. Similarly our program distinguishes pure earned premium, which provides coverage, from statement earned premium, which includes the gain or loss from cancellations, that is, the difference between UPR released by cancellation and refunds paid.

The function `ProjMonths` relies entirely on the transaction dates `pIssD`, `cBrkD`, `cPayD`, etc., in the data. These are not always the dates on which the transactions are recorded in an underwriter's books, because of processing lags above and beyond the report and payment lags reflected in the transaction dates. For this reason, and because of differences in the treatment of IBNR and RBNP reserves, the "statement" values produced by `ProjMonths` may differ slightly from values carried on the underwriter's statements, but for the most part they will actually give a clearer and less noisy picture of emerging results.

The last six lines of each table returned by `ProjMonths` show ratios useful in evaluating underwriting performance. These are (a) the loss ratio to pure earned premium, (b) the loss ratio to statement earned premium, (c) the ratio of refunds to UPR released by cancellation (a *refund ratio* analogous to the loss ratio), (d) the ratio of refunds to premium cancelled, (e) the fraction of total *UPR consumed* (earned premium plus UPR released) represented by earned premium, and (f) the ratio of losses plus refunds to UPR consumed, which we call the *payout ratio*. This last ratio plays a role similar to that of the loss ratio in more conventional lines of business, in that it may be compared with a "permissible payout ratio" to see, roughly, if the business is meeting underwriting targets.

If we let W = written premium, E = earned premium, U = UPR released by cancellation, L = losses, and R = refunds, $A = L/E$ (the loss ratio), $B = R/U$ (refund ratio), $C = E/(E+U)$ (earnings as a fraction of UPR consumed) , and $D = (L+R)/(E+U)$ (the payout ratio), then we have $D = AC+B(1-C)$. Moreover, if we observe a block of business to ultimate, we have $W=E+U$.

To show results at a higher level than calendar month by calendar month, we have included the function `ProjYears`, which converts the results of `ProjMonths` to years, and `ProjIDTUlt`, which converts the results of `ProjMonths` to a table with two numeric columns, one for

experience from inception through the latest transaction date, and one for experience from inception to "ultimate", here taken to mean through the latest projected month.

The following is an example of the output of the `ProjYears`. Only the first few columns are shown; the years actually run out through 2025. The corresponding run of `ProjMonths` has about twelve times as many columns, but otherwise looks very similar.

All terms, all issue months	200812	200912	201012	201112	201212
Pure written premium	63219048	55494635	68156208	86677470	102599334
Cancelled premium	3931840	6653499	9139261	12762231	16279208
Refunded premium	3637708	5400463	6919761	9352971	11696813
Beginning carried UPR	0	55598214	94303766	136961569	185670040
UPR released by cancellation	3771017	6038244	8163337	11339140	14422622
Pure earned premium	3849817	10750840	17335067	26629859	38004583
Ending carried UPR	55598214	94303766	136961569	185670040	235842169
Statement written premium	59581340	50094172	61236447	77324498	90902521
Statement earned premium	3983126	11388621	18578643	28616028	40730392
Paid Losses	1973280	6691022	11212481	15263851	20525493
RBNP reserve	0	0	0	0	0
Reported losses	1973280	6691022	11212481	15263851	20525493
IBNR reserve	379786	719661	906836	1146360	1267492
Incurred losses	2353066	7030897	11399656	15503375	20646625
Reserve for future refunds	5786503	8118597	10500894	13294115	16238998
SSAP 65 Test 1	50578187	77965513	106558250	139794532	174603520
SSAP 65 Test 2	54537977	90765525	129964951	174587357	220422516
SSAP 65 Test 3	25508895	42780093	61125338	83248803	106734510
Losses / pure EP (all ratios *100)	61.12	65.40	65.76	58.22	54.33
Losses / statement EP	59.08	61.74	61.36	54.18	50.69
Refunds / UPR released	96.46	89.44	84.77	82.48	81.10
Refunds / premium cancelled	92.52	81.17	75.71	73.29	71.85
EP / (EP+UPR released)	50.52	64.03	67.98	70.14	72.49
(Losses+Refunds) / (EP+UPR released)	78.61	74.04	71.85	65.46	61.69

In this example the paid and reported losses are identical because there are no report dates, distinct from payment dates, available in the data. Therefore the RBNP (reported but not paid) reserve is zero and the reserve listed as IBNR is actually an IBNP reserve, i.e. the sum of IBNR and RBNP. Also note that, for example for 2012, (loss ratio)(EP fraction)+(refund ratio)(1-EP fraction) = (0.5433*0.7249)+0.8110*(1-0.7249) = 0.6169, equal to the payout ratio, as expected.

Here the carried UPR was by the formula pro-rata after manufacturer's warranty (`PRafterMW`), while the earnings pattern used to project the losses was by experience. The carried UPR is the

more conservative, as may be observed by comparing the ending carried UPR for each year with SSAP 65 Test 2.

The following output from `ProjITDUlt` was derived from the same original run of `ProjMonths` as the example above. Here we are well beyond the expiration of all contracts issued on or before the requested latest issue date, and we assume no new business after that date, so all of the reserve items are zero in the Ultimate column. The various ratios, particularly the loss, refund, and payout ratios, give a good picture of the pricing adequacy of the business as written.

All terms, all issue months	ITD 201512	Ult 202512
Pure written premium	789927933	789927933
Cancelled premium	127140022	175631491
Refunded premium	92520293	116099209
Beginning carried UPR	0	0
UPR released by cancellation	112978623	150699868
Pure earned premium	280214714	639228065
Ending carried UPR	396734596	0
Statement written premium	697407640	673828724
Statement earned premium	300673044	673828724
Paid Losses	164097792	341879377
RBNP reserve	0	0
Reported losses	164097792	341879377
IBNR reserve	2467298	0
Incurred losses	166565091	341879377
Reserve for future refunds	25262641	0
SSAP 65 Test 1	288661351	0
SSAP 65 Test 2	366037003	0
SSAP 65 Test 3	175314286	0
Losses / pure EP (all ratios *100)	59.44	53.48
Losses / statement EP	55.4	50.74
Refunds / UPR released	81.89	77.04
Refunds / premium cancelled	72.77	66.1
EP / (EP+UPR released)	71.27	80.92
(Losses+Refunds) / (EP+UPR released)	65.89	57.98

The table shown is for the sum of all term groups and all cohorts; the analysis was conducted separately for four term groups and each was then separated into four cohorts; the output of the program includes tables for all 25 combinations of term groups and cohorts, including aggregates.

In addition to the tables shown above these programs return numeric vectors of inforce survival factors, refund/cancelled factors, carried and expected UPR factors, report and payment lag factors, and expected loss ratios, all separately by term group, and the following set of footnotes that the user may insert to explain whatever term group – cohort tables he or she chooses to print:

Earliest issue date 20080101; latest issue date 20151231; latest transaction date 20151231
Filter 1
Contracts 1292092; usable 1292091; claims 300820; usable 297849
Carried UPR by PRafterMW; expected earnings by experience
Function call: ProjITDUlt LA ProjMonths RA
LA= (<120),(<201204 201304 201404),(<12;0),(<<;._1 ' PRafterMW'),<''
RA= 20080101 20151231 20151231;'1';24 36 48
Working Directory C:/users/RLVaughan/documents/Test Data 4
Script version 20180712
Start 2018-07-12 17:42:36.951; end 2018-07-12 17:47:10.389; elapsed 0d 0h 4m 33.4380188s

The reader may wish to use `ProjMonths` as the starting point for customized reports leading in many possible directions. For example, a measure of the equity embedded in a company's reserves may be obtained as the difference between the UPR plus loss reserves shown on the company's books and the sum of SSAP 65 Test 3 and the IBNR, RBNP, and refund reserves shown by `ProjMonths`. For another example, if `ProjMonths` is run with contract counts substituted for premium and claim counts for losses, we obtain frequency ratios, e.g. claim counts to pure earned contracts; if it is run with contract counts substituted for premium but with normal losses we obtain pure premium ratios, and so forth.

3.8. Reports Involving Simulated Data

Our library includes a small group of functions to prepare reports comparing loss development methods applied to simulated, or synthetic, data. We include here only functions that *use* simulated data, not those that *generate* it. We refer the reader interested in creating sets of synthetic data to the simulator created a few years ago by the CAS Loss Development Simulation Working Party and available online. Written in R, this simulator is capable of generating entire loss histories in considerable detail, and it is thoroughly tested and documented. From whatever source, we assume that the user is in possession of a set of synthetic data, which will usually have the following general properties:

- It will consist of a number N of sample points, each of which is, or is derived from, a complete loss history generated stochastically from the same initial conditions (exposures,

time period observed, underlying distributions). Our programs should work comfortably, on most home or office computers, with a sample size N of, say, 10000.

- Each sample point, or loss history, may have originated in full detail captured via individual transaction records. However, for our purposes we assume that the records have been tabulated into triangles of the types expected by our standard loss projection programs described in Section 3.5 above. For simplicity we assume constant exposure.

- All simulated triangles are completed, with both "known" and "future" values, so they are really rectangles. The completed rectangles may include tail columns beyond the last known lag.

For concreteness we take the time periods and lags to be measured in years. This affects the labeling of our printed reports, not their structure, so our functions will return perfectly usable statistics for data compiled by months, quarters, halves, etc.

We are normally interested in comparing the performance of two or more reserve estimators by applying them to each sample point of the synthetic data and calculating global statistics such as means, standard errors, and correlations. We are not usually concerned with individual sample points, although occasionally an estimator will fail to run with a given sample point, or will run but produce anomalous or outlying results, leading us to study that sample point in detail to see how the estimator might be made more robust.

The function `SimTests` assumes that the user has coaxed the simulated data into a four-dimensional array `y` organized as (sample point) x (triangle type) x (time period) x (lag). By triangle type we mean the four triangles required by our loss projection programs (paid and incurred losses and claim counts by accident period, and paid losses by report period) but all of these in matrix form only (`SimTests` will supply the remaining parameters) and including both "known" and "future" values. If N=10000, m = 10 (the number of accident years), and n = 13 (the number of development lags to settle all simulated losses), then `y` will have shape `10000 4 10 13`. The "known" part of `y` may then have length k = 10, 11, 12, or 13 on its last dimension, since it is possible for writing to have ceased after 10 years but experience to have been tracked for one or more later years before being projected. For this reason, one of the parameters of `SimTests` tells it how many of the columns of each sample point in `y` are known, and the "known" triangles passed to the estimators will have this number of columns.

It is possible that some estimators presented with, say, 10 x 10 triangles will project these with tails to a development lag greater than 10, while other estimators will not project any tail. The user

may request that the system ignore the tail columns in the simulated runoff when comparing estimators of the latter kind. By default the tail runoffs *are* included, so part of the reported bias may be (justifiably) attributed to runoffs beyond the last lag in the known data.

A run of `SimTests` comparing paid with incurred Bornhuetter-Ferguson reserve projections produces (among other things) a formatted report, which looks like this:

Summary of Projected versus Simulated Loss Development

Estimator A. PaidBF (default parameters)

Year	ITD Paid	Est Ult	Sim Ult	Reserve	Runoff	Bias	RMS Err
1	452366	452366	452366	0	0	0	0
2	452341	452341	452341	0	0	-1	100
3	441742	441755	441761	13	19	-0.32623	69.77896
4	451038	451160	451193	123	155	-0.21170	31.14116
5	443203	444151	444173	948	970	-0.02285	11.34542
6	442875	448703	448647	5828	5772	0.00978	4.05110
7	419747	449051	449283	29303	29535	-0.00785	2.09997
8	344317	453186	453542	108868	109225	-0.00326	1.00359
9	176389	453297	453510	276909	277122	-0.00077	0.67179
10	21728	449768	448659	428039	426931	0.00260	0.50388
Total	3645746	4495778	4495476	850032	849729	0.00036	0.39784

Number of usable sample points: 10000

Estimator B. IncdBF (default parameters)

Year	ITD Paid	Est Ult	Sim Ult	Reserve	Runoff	Bias	RMS Err
1	452366	452366	452366	0	0	0	0
2	452341	452341	452341	0	0	-0.00561	0.56101
3	441742	441758	441761	16	19	-0.14151	9.87152
4	451038	451185	451193	147	155	-0.05268	5.36135
5	443203	444158	444173	955	970	-0.01551	2.45829
6	442875	448771	448647	5896	5772	0.02160	0.82297
7	419747	449282	449283	29535	29535	-0.00001	0.52309
8	344317	453544	453542	109227	109225	0.00002	0.34508
9	176389	453850	453510	277461	277122	0.00123	0.35860
10	21728	449068	448659	427340	426931	0.00096	0.45922
Total	3645746	4496324	4495476	850578	849729	0.00100	0.26919

Number of usable sample points: 10000

Maximum known lag 10; runoff lag 13; runoff lag included 13

Start 2018-07-18 09:46:19.003; end 2018-07-18 09:48:28.904; elapsed 0d 0h 2m 9.901000977s

`SimTests` produces additional results, including tables of correlation coefficients for each pair of estimators, separately for reserves and for prediction errors.

3.9. Trends

Trends are important in interpreting historical experience as well as in projecting future experience. It is often convenient to estimate historical trend internally or from exogenous sources such as industry data, and to select future trend by judgment with reference to historical trend. For simplicity future trend may be assumed equal to recent historical trend, and recent historical trend

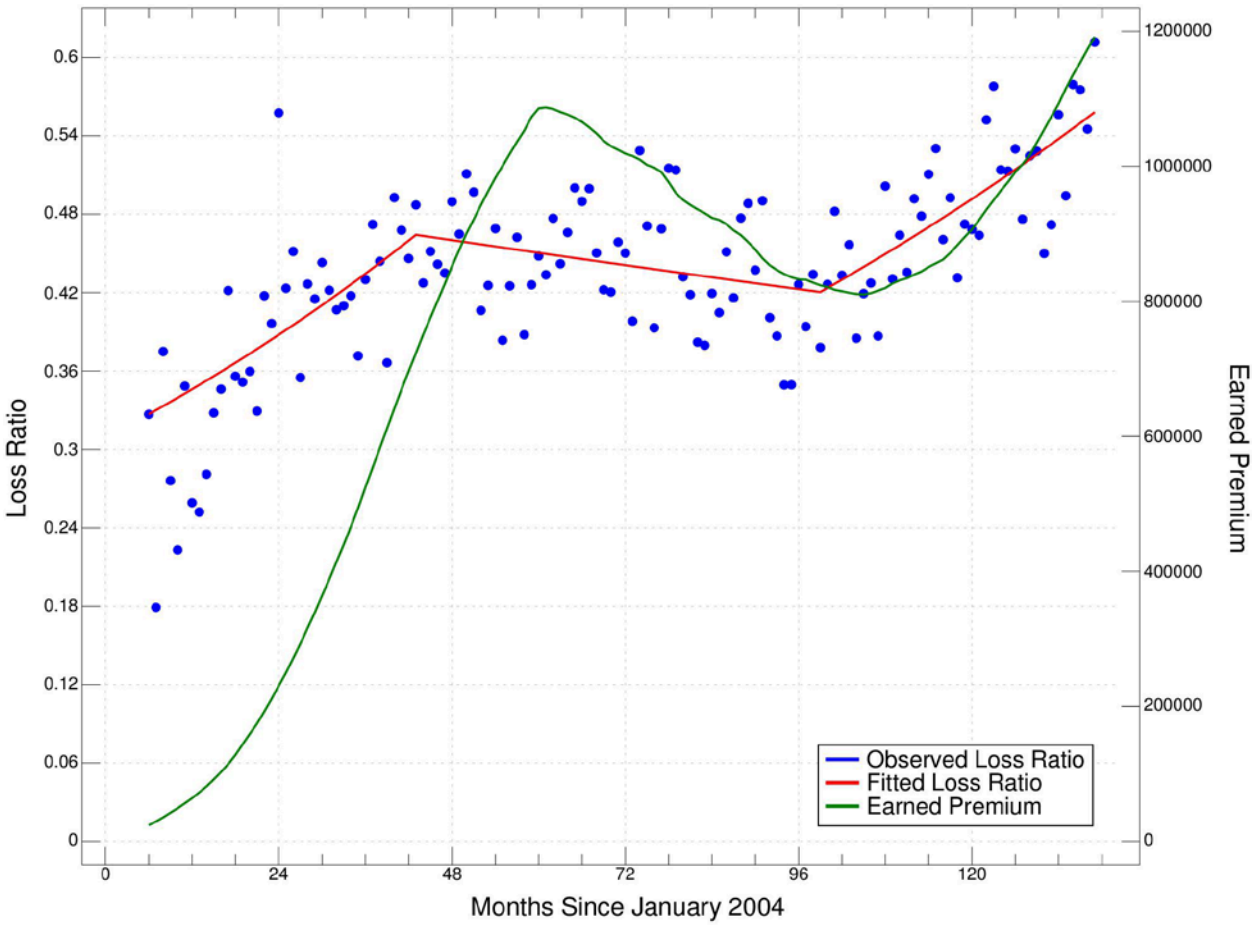
may be assumed constant over the observation period, though either assumption may prove unwarranted in particular cases.

The library provides several functions dealing with trend. The function `ExpTrend` fits an exponential trend to the ratios of an "amount" variable to an "exposure" variable, both dependent on a "time" variable, using weighted nonlinear least squares. This is invoked by `LRTrend` to estimate trends in losses per earned premium (loss ratios), losses per earned contract (loss costs), and claims per earned contract (frequency), and by `SevTrend` to estimate trend in losses per claim (severity), all with respect to the number of months since a given date.

Example. The plot below shows loss ratios estimated from experience for Warranty contracts issued from January 2006 through December 2015, with actual term 57 through 66 months in our anonymous data set (another cluster from the example in the previous section), but with the first and last six months dropped to avoid introducing outliers based on immature or sparse data.

The plot shows earned premium as well as loss ratios and fitted loss ratios. The earned premium starts out very low in the early months of the block of business studied (because of its immaturity), increases to the beginning of 2009, decreases until late 2012, and increases thereafter, apparently reflecting, with some delay, the effect of the recession that began in 2008. Inspection of the loss ratios (the dot plot) suggests that they follow a similar pattern, so we set the program `LRTrend` to look for two change points, producing the three-part fitted curve shown below.

It is not a good idea to look for too many change points, as this may pick up noise rather than signal, and the search for the best combination may become unreasonably long. In the present case, however, the changing trends are obvious enough to warrant fitting them and looking for concrete explanations, such as changes in economic conditions or changes in plan design, underwriting, or claims handling, at approximately the dates indicated. It is interesting to note that the loss ratios responded to these changes with less delay than the earned premium, though probably still with more delay than the written premium (not shown). The segments between change points are not straight lines but are exponential curves, and the programs give the intercept and trend for each.



3.10. Gibbs Sampling

Gibbs Sampling is a variant of Markov Chain Monte Carlo (MCMC) estimation particularly well suited to Bayesian networks: multivariate probability distributions representable as directed acyclic graphs, where the conditional distribution at each node is known, but the full joint distribution may be very difficult to calculate. There are several Gibbs sampling programs in the public domain. The best known of these are the BUGS family ("Bayesian inference Using Gibbs Sampling"; this includes WinBUGS and OpenBUGS, which use essentially the same language and graphical user interface. Our library includes a few examples showing how J may interact with BUGS.

Since Gibbs sampling can estimate the entire posterior distributions of the variables in a model, individually and jointly, applications to the study of the variability of loss reserves, conditional on triangle data usually available, come readily to mind. It turns out that when the loss process is modeled chronologically, for example with latent variables representing the occurrence of losses, distributed multinomially to report and then to payment periods, or when the observed data is cumulative, it is awkward to express the model in BUGS. But rearranged and simplified models of the loss process are feasible. The function **EELSBUGS** (where EELS stands for "**E**xposure * **E**LR

* **L**ag factor * average **S**everity) is a start (and only a start) in this direction. This function models *claims* in each accident year x payment lag cell with a Poisson, Gamma-Poisson, or normal distribution, with mean based on "EEL"; severity is either taken as a constant average amount, to rescale the coefficient of variation of *losses* to that of the claims, or is distributed normally (censored below at 0) with prior distributions for the parameters μ and $\tau = 1 / \sigma^2$ (precision) used by BUGS.

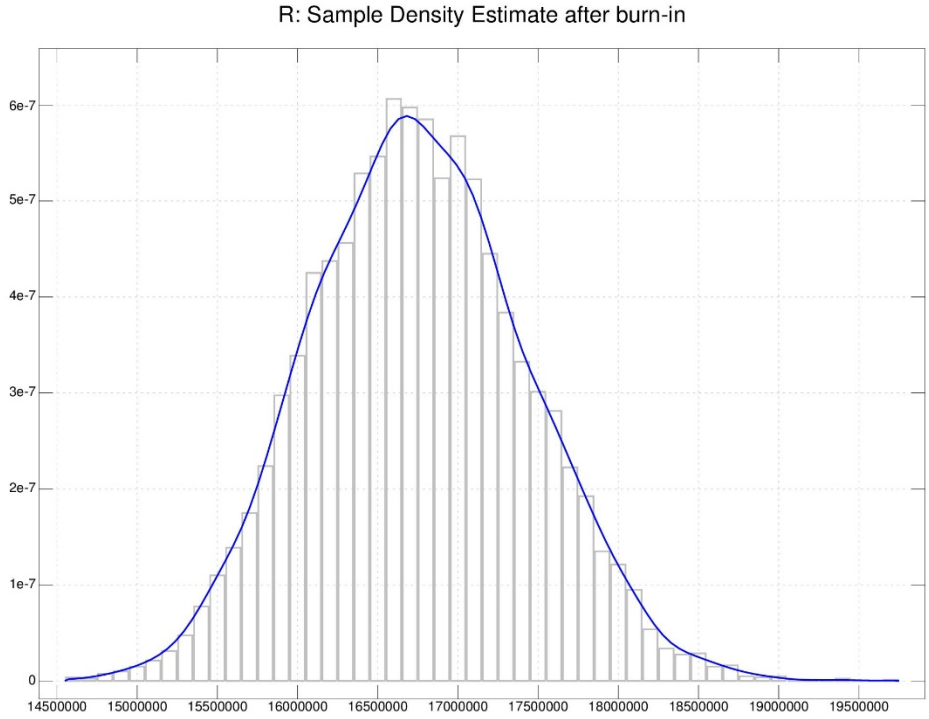
We also include the functions `CPtsG`, which locates change points in the vector `y` by Gibbs sampling performed directly in J, and `CPtsBUGS`, which does the same thing through BUGS, but limited to a single change point. These functions are not as fast as the function `FindCPts`, which finds change points by least squares, but they yield the complete posterior distributions of the lengths, and of the means, of the subvectors defined by the change points, and they replicate an example available in an online tutorial on Gibbs Sampling.

Our functions separate the generation of a Gibbs sample from its analysis. The sampling is done without a separate burn-in period and without thinning: these steps are deferred until the analysis, by the function `PostG`, which selects the sample representing the final distribution and produces much clearer and more detailed graphical output than BUGS itself. The conjunction `DoBUGS`

combines the steps, running a program `v` of the form of `CPtsBUGS` or `EELSBUGS`, with arguments `x` and `y`, followed by `PostG`, with left argument `m` . For example, executing the line

```
D=.'' DoBUGS EELSBUGS TaylorAshe;''
```

runs `EELSBUGS` with default arguments (a simple claim count model assuming average severity) and uses `PlotG` to generate statistics and plots, such as the following histogram and kernel density plot of the reserve:



Here the result `D` includes the BUGS details from `EELSBUGS` and the sample details from `PlotG`, and the plots include sample histories, distributions, and autocorrelations.

3.11. Life Contingencies

The packages available with J include one, finance/actuarial, that is specifically devoted to life contingencies (the valuation of reserves and premiums for annuities and insurances). These functions are polished and include many useful details, such as deferral and guarantee periods, and interpolation for off-anniversary ages. They are found in **interest.ijs** in the subdirectory finance, and in **actfns.ijs**, **actfnsm.ijs**, and **actutil.ijs** in the subdirectory actuarial, all (on Windows systems) in the directory **Program Files/J64-806/addons**. Please note that, as of early 2018, these functions can only be loaded by direct reference to their complete pathnames, and that the life tables (qx) available in the file **actables.ijf** cannot be accessed. However, if a q_x vector is supplied from another source, the functions work fine. Moreover, the Society of Actuaries has put online an

enormous library of mortality tables along with an annuity calculator with a customized user interface.

With these excellent tools available it would be pointless to replicate their functionality here. But it is remarkable how well life contingencies illustrate the power and conciseness of J. For this reason, and as a nod to the Life side of the actuarial world, we include a few small functions in our library. For example, `AnnD1L` calculates the values of a single-life annuity due at exact ages, where `x` is a mortality table, as the vector q_x, and `y` is (age, term, interest rate). `AnnDJS` calculates the value of a joint-and-survivor annuity due where `x` is the vector q_x and `y` is (age of first annuitant, age of second annuitant, term, interest rate). `AnnI1L` and `AnnIJS` do the same for annuities immediate. For purposes of illustration only, we include in our script the vectors `SS2014M` and `SS2014F` of probabilities q_x for the male and female Social Security Period Life Tables for 2014. We recognize that no-one would price an annuity, and few would price an insurance, based on these tables, but they happened to be available online and they are perfectly useful as illustrations. They run from age 0 through age 119.

3.12. Miscellaneous

As an illustration of programming technique our library includes a function to solve Sudoku puzzles. We recognize that many actuaries traveling to CAS meetings find it impossible to see such a puzzle in the airline's magazine without solving it. The program `Sudoku` will let you perform this important task and satisfy yourself that the solution is unique (a criterion of good design), without spoiling the puzzle for the next passenger in your seat. And, as it takes but a few seconds to enter and run a puzzle, you will save time for more valuable activities such as reading e-Forum articles. Execute `Sudoku SudExample0` to see a sample solution.

Along the same lines is the program `PlayCGL` which runs instances of J.H. Conway's Game of Life. To see an example, execute `zz=.30 40 15 20 30 PlayCGL CGLExample3` (or just `zz=.PlayCGL CGLExample3`, which does the same thing but sets up the display a bit less elegantly). The game is described in the comments to `PlayCGL`. Note that the core calculations of this game are handled by three very concise tacit functions, one of which contains the loopless code that searches for populated neighboring cells. Also note the convenient way that J allows us to maintain a cache of previous states to search for cycles.

We have also included, in the script **rubik.ijs**, functions to generate and solve Rubik's Cube puzzles, which intrigue many actuaries. These functions provide examples of the management of permutation vectors. They are in a separate script to make their inclusion optional, and the script puts its functions in a separate locale, `rubik`, to avoid cluttering the base namespace. The

functions in **rubik.ijs** require **act.ijs** to be present, and **rubik.ijs** defines links, in the base locale, to a few top-level functions. These functions generate cube puzzles and then solve them, systematically but with no pretense whatever to optimal length. To generate and solve a puzzle you may run, for example, `SolveCube >1{RandCube 20`; to watch a solution unfold move by move, run `ShowSolution >1{RandCube 20`.

3.13. Further Development

The library as it stands is a collection of building blocks leading to higher-level functions for a few actuarial procedures. The reader may find some of these functions immediately useful without extension or modification. But he or she will only realize the full potential of the language J, and of the library as a starting point, by using it to build new functions to solve new problems. Many of these functions will, most likely, generate reports unique to a specific company or client. We suggest the following general approach:

- Maintain the script **act.ijs** as downloaded from the CAS site, and code new functions in new scripts. Load **act.ijs** first, then the new scripts. In this way you will be able to install new versions of **act.ijs** (which may appear in the future) without affecting your own work.
- We suggest creating one new script for general-purpose tools (such as, say, a GLM system) and another script for tools customized to particular clients (such as a specialized report).
- Whenever an overall assignment requires application of several library programs, consider writing a cover function to apply them all in the proper order and with the proper arguments. This will greatly reduce the possibility of errors in following the procedure. If we had expected to need to repeat the process, we could have done this in the example for Section 3.6 above, to make sure we did not omit or mis-handle any of the eight steps required.
- In a large report, include footnotes documenting the function used to generate it, the version dates of any scripts loaded, and, if small, the left and right arguments of the function. This will permit later replication of the report, and it will facilitate updating of the report for new data. If the arguments are large, they may be converted to a linear text format using the function `5!:5` (see the documentation of the foreign conjunction `!:`) and then saved to text files.
- If a program saves results to file, consider attaching a timestamp to the filename to avoid overwriting earlier files.

- If a program saves multiple results to file (e.g. a report and a plot, or a report and linearized versions of its arguments), consider creating a new folder (under program control) and placing all of the multiple results in that folder. The name of the folder might include a timestamp but the names of the files need not.
- It is possible to use locales to compartmentalize work into classes, objects, and methods. We have not done so with **act.ijs** as we felt it would be easiest for the user to have all tools at hand in the base locale when programming new functions. But you may prefer to use locales to isolate lower-level functions supporting top-level public applications, thereby avoid cluttering the namespace.
- Try to think of each assignment as a special case of a class of assignments, and write your functions to address the class rather than the special case. When satisfied with the results for the case at hand, polish the code to produce similarly satisfactory results for other instances of the general class, and save the polished code for future use.

4. CONCLUSIONS

We hope we have demonstrated the power and elegance of the J language and its remarkable ability to express actuarial algorithms that use numeric arrays. We hope that our tutorial will help the reader get started learning J, though no tutorial however thorough will substitute for hands-on experimentation. We hope that the library functions will serve both as solutions to some problems in their own right, and as examples of J technique, or a stimulus to the reader to improve upon our J technique. Above all, we hope that the reader will extend our library with a collection of new solutions to his or her new problems … and will share some of these solutions with the wider actuarial community.

REFERENCES

[1] Agbeko, Tony, Munir Hiabu, Maria Dolores Martinez-Miranda, Jens Perch Nielsen, and Richard Verrall, Validating the Double Chain Ladder Stochastic Claims Reserving Method, *Variance* Volume 8 Issue 2, 2014: 138-160.

[2] Belhadi, Hassine, Vincent Goulet, and Tommy Ouellet, On Parameter Estimation in Hierarchical Credibility, *ASTIN Bulletin*, Volume 39, Issue 2, November 2009: 495-514.

[3] Bornhuetter, Ronald and Ronald Ferguson, The Actuary and IBNR, *PCAS* LIX, 1972: 181-195.

[4] Bühlmann, Hans, Vereinigung Schweizerischer Versicherungsmathematiker / Association des Actuaires Suisses, Ecole d'été 1983, Estimation of IBNR Reserves by the Methods Chain Ladder, Cape Cod, and Complementary Loss Ratio, unpublished.

[5] Dutang, Christophe, Vincent Goulet, Xavier Milhaud, and Mathieu Pigeon, Credibility Theory Features of Actuar, https://cran.r-project.org/web/packages/actuar/vignettes/credibility.pdf, 2008.

[6] Gluck, Spencer M., Balancing Development and Trend in Loss Reserve Analysis, PCAS LXXXIV, 1997: 482-532.

[7] Kerper, John and Lee Bowron, An Exposure-Based Approach to Automobile Warranty Ratemaking and Reserving, CAS Forum 2007: 29-43.

[8] Mack, Thomas, Distribution-Free Calculation of the Standard Error of Chain Ladder Reserve Estimates, ASTIN Bulletin 23 (1993), 213-225.

[9] Merz, Michael and Mario V. Wüthrich, Paid-incurred chain claims reserving method, *Insurance: Mathematics and Economics*, 46, 2010: 568-579.

[10] Müller, Thomas, Projection for Claims Triangles by Affine Age-to-Age Development, *Variance* Volume 10, Issue 1: 121-144.

[11] Quarg, Gerhard and Thomas Mack, Munich Chain Ladder: A Reserving Method that Reduces the Gap between IBNR Projections Based on Paid Losses and IBNR Projections Based on Incurred Losses, *Blätter der Deutschen Gesellschaft für Versicherungs- und Finanzmathematik*, volume 26, number 4, 2004: 597-630. Reprinted in *Variance*, Volume 2/Issue 2, 2008: 266-299.

[12] Rich, Henry, J for C Programmers, www.jsoftware.com (accessible through help menu of J interpreter).

[13] Vaughan, Richard, The Unearned Premium Reserve for Warranty Insurance, CAS E-Forum, Fall 2014.

[14] Vaughan, Richard, Residual Loss Development and the UPR, CAS E-Forum, Spring 2017.

[15] Yamashiro, Marcus M., Recursive Credibility: Using Credibility to Blend Reserve Assumptions, *Variance* Volume 8 Issue 2, 2014: 105-137.