Estimation of value at risk using Johnson's S_U-normal distribution
P. Choi (2001)

Value at risk (VaR) estimation can be univariate (you only have the returns of a portfolio) or multivariate (you know the returns of the constituents of your portfolio, *i.e.*, you want to estimate a quantile of $\sum_i w_{it} r_{it}$, where r and perhaps also w is a random variable); it can be conditional (e.g., conditional heteroskedasticity, but you could also have models with conditional third or fourth moments; the article also mentions exponential smoothing (RiskMetrics), which is a special case of IGARCH) or unconditional (extreme value theory (EVT)).

The article studies VaR estimation in a GARCH model with S_U-normal innovations.

A random variable Y is S_U-normal if it is of the form

$$Y = \sinh(\lambda + \theta X)$$

where $\theta > 0$ and X is standard gaussian. Those distributions can be skewed and have fat tails.

Who needs hedge funds? A copula-based approach to hedge fund return replication
H.M. Kat and H.P. Palaro (2005)

Hedge fund replication usually tries to mimic the distribution of the returns of a hedge fund, but hedge funds are often used as an overlay to an already existing portfolio: the authors suggest to also replicate the dependency between the hedge fund and the existing portfolio, as follows.

- Infer the joint distribution of the hedge fund and the existing portfolio using copulas (gaussian, Student, Gumbel, Cook-Johnson (aka Clayton), Frank, symmetrised Joe-Clayton (SJC)) and marginal distributions (gaussian, Student, Johnson S_U);
- Among all the payoff functions, find the cheapest that produces this distribution (this is Dybvig's *payoff distribution pricing model* (PDPM) generalized to a 2-dimensional payoff distribution); it suffices to consider path-independent payoff function (any payoff distribution generated by a path-dependent payoff function can also be generated by a path-independent payoff funcion); if the Sharpe ratio of the underlying asset is high enough and the correlation with the investor's portfolio low enough, the payoff function will be non-decreasing;
- Price and find a replicating strategy for this payoff function.

Reinforcement learning: a survey
L.P. Kaelbling et al.
Journal of artificial intelligence research (1996)

In *reinforcement learning*, the agent has to make choices and receives some feedback on those choices. Contrary to supervised learning, he is not told which action would have been the best; furthermore, the consequences of an action need not be immediate.

The agent typically tries to maximize one of the following quantities (they do not lead to the same optimal policies):

– N-step optimal control

$$E\left[\sum_{t=n+1}^{N} r_t\right], \qquad n \in [\![0, N]\!]$$

– N-step receding horizon control

$$E\left[\sum_{t=n+1}^{n+N} r_t\right]$$

– Infinite-horizon discounted model

$$E\left[\sum_{t=1}^{\infty} \gamma^t r_t\right]$$

– average reward model (this one does not penalize for long learning times)

$$\lim_{N\to\infty} E\left[\frac{1}{N}\sum_{t=1}^{N} r_t\right]$$

Asymptotic convergence results are useless: a fast convergence to a near-optimal solution is better than a sluggish convergence to the optimal.

Algorithms have to make a trade-off between exploration (of the possible choices and their consequences) and exploitation.

Dynamic programming can be used to maximize those performance measures.

With a 1-step, immediate-reward model, one can use greedy strategies, *Boltzman exploration* or *interval-based techniques* (for each action, store the number of trials and the number of successes and compute a confidence interval on the probability of success). These algorithms can be adapted to multi-step, immediate-reward problems.

Markov decision processes (MDP) can be used to model delayed reward. If the decision process were known (we do assume that the states and transition probabilities are known, though), we could compute the value of each state for the discounted model and, from there, the optimal policy.

Value iteration estimates the value of each node and derives a strategy (since a strategy is a finite object, it converges in a finite number of steps); *policy iteration* directly estimates the policy.

But we do not know the model.

The *adaptive heuristic critic* algorithm is similar to policy iteration, but the value function is computed iteratively, from the $TD(0)$ algorithm:

$$V(s) \leftarrow V(s) + \alpha(r + \gamma V(s') - V(s))$$

where r is the reward, s the current state, s' the next state. The $TD(\lambda)$ algorithm also updates states that were recently visited.

The *Q-learning* algorithm (if you do not know which algorithm to implement, use this one) estimates $Q(s, a)$, the expected discounted reinforcement of taking action a while in state s, with the $TD(0)$ or $TD(\lambda)$ algorithm, with a decreasing α. You must first explore the states (but the algorithm is robust) and, after convergence, you can act greedily.

Other algorithms (*certainty equivalence*, *Dyna*, *prioritized sweeping* learn the model at the same time as the optimal policy: they make a better use of the data and converge faster.

In case of very large state spaces, you can aggregate some states and solve a coarser problem (*multi-grid methods*, *state aggregation*); you can also replace the mappings (transition probabilities, value functions, policies, etc.) by a simpler representation obtained by *supervised learning*.There are other *generalization* algorithms.

Design of an FX trading system using adaptive reinforcement learning
M.A.H. Dempster
Carisma (2007)

Recurrent reinforcement learning (RRL), *i.e.*, a recurrent 1-layer neural network, can be used to turn returns time series into a buy/sell/wait order, so as to maximize a *moving-average Sharpe ratio*,

$$\frac{\text{EWMA(returns)}}{\text{EWMA(returns}^2\text{)}}.$$

This used to work, one decade ago. One can try to improve the model as follows:

– Do not only consider past returns: add in technical indicators – actually, this does not add anything: RRL already extracts all the relevant information;
– Replace the transaction costs parameter by something larger than the bid-ask spread, so as to favour trades with larger returns;
– Fix the instability in the neural network weights by shrinking them;
– Update the weights twice at each step, *i.e.*, replace $w_t = f(x_t, w_{t-1})$ with $w_t = f(x_t, f(x_t, w_{t-1}))$.
– Add a risk and performance management layer in the algorithm, with a stop-loss and a shutdown procedure (to stop investing when the algorithm becomes unstable after a regime switch); the suggested risk measure takes into account both total loss and the size of the individual losses (this is similar to

Omega):

$$\Sigma = \frac{\sum {r_{i-}}^2}{\sum {r_{i+}}^2};$$

– Update the neural network weights at each step, but do not update the meta-parameters (transaction cost, trading threshold, etc. – there are five of them) that often;

The following improvements have not been tested yet:

– Add other information, such as the order flow or the limit order book;
– Use the algorithm on several currency pairs; generalize the risk control accordingly;
– Apply the algorithm to market making instead of trading.

Computer science and game theory
J.Y. Halpern
`arXiv:cs/0703148`

Complexity plays a role in game theory:

– Agents can have *bounded rationality*, *i.e.*, have limited computational power (e.g., they can be finite automata); this can lead to cooperation in the prisoner's dilemma;
– Many problems about *Nash equilibrium* are NP-hard;
– One cannot design a non-dictatorial voting scheme immune to manipulation by voters, but this manipulation can be made computationnally unreasonable; there are similar problems in *combinatorial auctions* (auctions where you can bet on bundles of objects; but pricing 2^N objects is too long);
– Agent-based games (e.g., *Byzantine agreement* are very similar to distributed computing, networking protocols, fault-tolerance, mediator-less cryptography.

This (concise) article also mentions random graphs, bayesian and Markov networks, reinforcement learning.

Forecasting with many predictors
J.H. Stock and M.W. Watson (2005)
Handbook of economic forecasting

The authors review several classes of forecasting algorithms: forecast combination (without any discussion of bagging or boosting), bayesian model averaging (BMA, rarely used, but better than an equal-weighted combination of predictors), *empirical Bayes methods* (the prior is not a prior, it comes from the data) and dynamic factor models (no mention of biased estimators or regularization paths).

In a *dynamic factor model* (DFM), unobserved AR factors $\mathbf{f}$ and their lags explain the observed variables X_{it}:

$$\mathbf{f}_t + \Gamma_1 \mathbf{f}_{t-1} + \cdots + \Gamma_k \mathbf{f}_{t-k} = \boldsymbol{\eta}_t$$
$$X_{it} = \lambda_{i0}\mathbf{f}_t + \cdots + \lambda_{i\ell}\mathbf{f}_{t-\ell} + u_t$$

while a (static) factor model would be

$$X_{it} = \lambda_i f_t + u_{it}.$$

Dynamic factor models can be reformulated as static factor models and estimated by maximum likelihood; there are also principal-component-analysis-based approximations.

The article interprets this model in terms of the *spectral density matrix* without bothering to define it.

Dynamic principal component analysis (PCA) is similar; the corresponding latent variables can be obtained by performing a PCA dimension reduction on the spectral density.

Principal components at work: the empirical analysis of monetary policy with large datasets
C.A. Favero et al. (2002)

Comparison of two estimators of *dynamic factor models*, based on static (time-domain) and dynamic (frequency-domain) PCA (principal component analysis): the performance is similar, but dynamic PCA is more parsimonious.

Moment problems via semi-definite programming: applications in probability and finance
P. Popescu and D. Bertsimas (2000)

The Markov, Chebychev, Chernoff inequalities provide bounds on some probabilities $P[X \in \Omega]$ given some moments or joint moments of arbitrary random variables. More generally, one can try to solve the following optimization problem:

$$\text{Max}\{\, E[\phi(X)] \;:\; \forall i \; E[f_i(X)] = q_i,\; X : \Omega \longrightarrow \mathbf{R}^m \,\}$$

For instance, in finance, one can look for bounds on the price of an option given the moments of the underlying and/or other option prices (we only have to assume that the option price is the expected discounted payoff under the risk-neutral measure; we do not know anything about that measure except that it is a probability measure).

The *feasibility* of that optimization problem can be expressed in terms of positive semi-definitiveness – e.g., in a multi-dimensional context, the second centered moment (the variance matrix) should be positive semi-definite.

In the *dual* problem (under reasonable conditions, there is a strong duality theorem), the expectation disappears but we have an inifite number of constraints:

$$\begin{aligned} &\text{Minimize } y'q \\ &\text{such that } \forall x \in \Omega \quad y'f(X) \geqslant \phi(x). \end{aligned}$$

Fortunately, those constraints can often be expressed as semi-definite constraints and the problem can be solved.

In higher dimensions, beyond the second moment, the problem of finding optimal bounds is NP-hard.

This is a review article: there are more detailed publications by the same authors.

Applications of second order cone programming
M.S. Lobo et al. (1998)

Many optimization problems can be recast as second order cone programs (SOCP): quadratically-constrained quadratic programs (QCQP), optimization problems in which the objective is a sum or max of norms; FIR (finite impulse response) filter design, robust linear programming, robust least squares, robust portfolio optimization.

Interior-point methods adapted to SOCP are faster than those adapted to the more general semidefinite programs (SDP).

Second-order cone programming
F. Alizadeh and D. Goldfarb (2002)

More detailed article on the same subject.

Second order cone programming approaches for handling missing and uncertain data
P.K. Shivaswamy et al.
Journal of machine learning research (2006)

In the support vector machine (SVM) binary classification optimization problem

$$\begin{aligned} &\text{Minimize } \tfrac{1}{2}\left\|z\right\|^2 + C\sum_i \xi_i \\ &\text{such that } \forall i \quad y_i(\langle w, x_i\rangle + b) \geqslant 1 - \xi_i \\ &\qquad\quad \forall i \quad \xi_i \geqslant 0 \end{aligned}$$

uncertainty can be introduced by transforming the constraints into

$$P\big[y_i(\langle w, x_i\rangle + b) \geqslant 1 - \xi_i\big] \geqslant 1 - \kappa_i.$$

The *Chebychev inequality* gives a worst-case bound on this probability, provided you know the first two moments of x.

This can be generalized to classification into more than two groups and to regression.

Incorporating estimation errors into portfolio selection: robust portfolio construction
S. Ceria and R.A. Stubbs
Journal of asset management (2006)

The the list of alternatives to mean-variance portfolio optimization:

- *James–Stein estimators* shrink the expected returns of each asset to the average expected returns, depending on the volatility of the asset;
- *Jorion estimators* shrink the expected returns estimate towards the minimum variance portfolio;
- The *Black–Litterman* model blends views on some portfolios to implied market returns;
- *Michaud's resampled portfolios* are computed as follows: on bootstrap samples, estimate average returns and variance matrix, compute the optimal portfolio, average those portfolios;
- Adding constraints might improve (or worsen) the sensitivity of the portfolio to parameter changes;
- Robust optimization, where we only give the optimizer intervals containing the alpha;

the authors add a new one.

One can estimate the error on the expected returns of the portfolio and maximize this return with a penalty for the estimation error:

$$\begin{aligned} &\text{Maximize} && \mathbf{w}'\boldsymbol{\alpha} - \lambda \left\|\Sigma^{1/2}\mathbf{w}\right\| \\ &\text{st} && \mathbf{w}'V\mathbf{w} \leqslant v \\ &&& \mathbf{w}'\mathbf{1} = 1 \\ &&& \mathbf{w} \geqslant 0 \end{aligned}$$

where λ is the penalty for the estimatio error, Σ is the variance matrix of the estimation errors, V is the variance matrix of the returns, $\boldsymbol{\alpha}$ are the forward returns, v is the risk target and $\mathbf{w}$ are the portfolio weights the optimizer should find.

This is not a quadratic problem, but a *second-order cone program*.

Stochastic programming models for asset liability management
R. Kouwenberg and S.A. Zenios (2001)
in *Handbook of asset and liability management*

Scenario trees for a stochastic program, in an asset liability management (ALM) context, can be generated with one of the following methods:

- random sampling from the model (you might want to trim dow the resulting tree, as in *importance sampling*);
- adjusted random sampling (use *antithetic sampling to fit every odd moment of the underlying distribution*);
- produce returns that match the first few moments of the distribution (this optimization probmen can be longer to solve than the final ALM problem).

You should make sure not to introduce arbitrage opportunities because of discretization or approximaation errors: the no-arbitrage condition can be added as a linear constraint.

Stochastic programming needs model generation tools – in particular, there is still no stochastic optimization language.

ALM can also be tackled with mean-variance optimization: just add a "liability" asset with a prescribed weight.

This chapter also defines a *myopic* investor: a multi-period investor who behaves as a 1-period investor, for instance because of his constant relative risk aversion (CRRA), if he has a power utility.

Improving investment performance for pension plans
J.M. Mulvey et al.
Journal of asset management (2006)

We do not live in a one-period world; traditional risk-reward mesures such as the Sharpe ratio are not directly applicable to a multi-period setup; one should consider multi-period strategies, stochastic programming, etc.

Managing guarantees
M.A.H. Dempster
Journal of portfolio management (2006)

The methods of asset-liability management (ALM), *i.e.*, dynamic stochastic programming, can be used to implement guaranteed strategies, thereby competing with portfolio insurance.

The technical details of the article might be "a challenge even for sophisticated users".

Extending algebraic modelling languages for stochastic programming
P. Valente et al.

It is easier to formulate optimization problems using a declarative language (AMPL, GAMS, AIMMS, MPL, to name a few) than with a list of huge matrices.

This article advocates the use of a similar language, SAMPL, for *stochastic programs*. It also reviews the various types of stochastic programs and gives a few examples (in particular, how to turn a multistage stochastic program into a deterministic one: expand all the scenarios and add non-anticipatory constraints).

SAMPL is implemented in SPInE (commercial).

Also check SMPS (not an algebraic language).

Composing contracts: an adventure in financial engineering
S. Peyton Jones et al.
Functional Pearl, ICFP (2000)

Haskell (or any other functional language) can be used to *describe* complicated financial contracts (options on options on... on options, with complicated cash flows) – the industry lack such a precise notation – and then to price them (but there are still optimizations to be made; and one might prefer to use C for the final computations).

Caml trading: experiences in functional programming on Wall Street
T. Minsky
The monad reader (2007)

Benefits of functional languages in finance.

Constructive homological algebra and applications
J. Rubio and F. Sergeraert (2006)

Another unlikely application of functional programming.

Data Envelopment Analysis
G.N. Gregoriou and J. Zhu
Journal of portfolio management (2007)

The notion of *efficient frontier* can be generalized to higher dimensions: several risk measures or inputs (standard deviation, downside deviation, maximum drawdown), several outputs (returns, proportion of profitable months, maximum consecutive gain).

If you have enough data points, they provide an approximation of the efficient frontier (?).

The *efficiency* (closeness to the efficient frontier) can be defined as the solution of a linear problem:

Minimize θ such that

$$\sum_k \lambda_k x_{ik} \leqslant \theta x_{ik_0}$$

$$\sum_k \lambda_k y_{jk} \geqslant y_{jk_0}$$

$$\sum_k \lambda_k = 1$$

$\lambda_k \geqslant 1$

i : input

j : output

k : funds

k_0 : fund whose efficiency is being computed

x_{ik} : input i of fund k

y_{jk} : output j of fund k.

See `http://people.brunel.ac.uk/~mastjjb/jeb/or/dea.html` for a picture.

Optimization of the largest US mutual funds using data envelopment analysis
G.N. Gregoriou
Journal of asset management (2006)

Another article on data envelopment analysis (DEA, sometimes also called *frontier analysis*). It defines the *classical efficiency* as

$$\mathrm{CCR}_{k_0} = \mathrm{Max}\left\{ \frac{\sum_j \mu_j y_{jk_0}}{\sum_i \lambda_i x_{ik_0}} \text{ st } \forall k \ \frac{\sum_j \mu_j y_{jk}}{\sum_i \lambda_i x_{ik}} \leqslant 1 \right\}$$

and the *super-efficiency* (which is no longer bounded by 1) as

$$\mathrm{CCR}_{k_0} = \mathrm{Max}\left\{ \frac{\sum_j \mu_j y_{jk_0}}{\sum_i \lambda_i x_{ik_0}} \text{ st } \forall k \neq k_0 \ \frac{\sum_j \mu_j y_{jk}}{\sum_i \lambda_i x_{ik}} \leqslant 1 \right\}.$$

They also mention the *cross-efficiency model*, but not clearly.

On the use of data envelopment analysis in assessing local and global performances of hedge funds
H. Nguyen-Thi-Thanh (2006)

Yet another article on the applications of data envelopment analysis (DEA) to hedge fund comparison – previous ones forgot the fees.

An empirical study of multi-objective algorithms for stock ranking
Y.L. Becker et al.

Genetic algorithms can be used to simultaneously optimize several goals: put the solutions on separate "islands", one for each goal, and have some migrate from time to time.

This is very similar to *data envelopment analysis* (DEA) but I do not expect the algorithm to converge towards a uniquely defined (or meaningful) solution; if the migration rate is well chosen, it should converge to a (set of) point(s) on the efficient frontier, not far away from the optimal solutions of the one-goal problems.

The algorithm is applied to a stock selection problem with the following goals: information ratio (IR), information coefficient (IC) and intra-fractile hit rate (IFHR), *i.e.*, proportion of stocks in the top (resp. bottom) decile that outperform (resp. underperform) the average.

Equilibrium underdiversification and the preference for skewness
T. Mitton and K. Vorkink
Review of financial studies (2007)

Preference for skewness, *i.e.*, maximization of

$$E[X] - \frac{1}{2\tau}\operatorname{Var} X + \frac{1}{3\phi}\operatorname{Skew} X,$$

as with "Lotto investors", explains underdiversified portfolios.

Do losses linger?
R. Garvey et al.
Journal of portfolio management (2007)

Traders who experienced a loss in the morning are more risk-seeking in the afternoon: this *disposition effect* is in agreement with *prospect theory*, defined here as the maximization of an s-shaped "utility", function of gains and losses instead of total wealth.

Stocks as lotteries: the implications of probability weighting for security prices
N. Barberis and M. Huang (2007)

Prospect theory differs from expected (concave) utility theory:

- The utility function is concave over gains and convex over losses; it has a kink (*i.e.*, left and right derivatives are different) at the origin;
- The investor does not maximize the expected utility but a weighted utility: the probabilities are transformed (but these are not subjective probabilities, just decision weights: the investor is perfectly aware of the objective probabilities); as a result, low probabilities are overweight: the investor wants both lottery and insurance.

Prospect theory is incompatible with first-order dominance, but can be modified into *cumulative prospect theory*, which applies the weights to the cumulative distribution function. Often, one chooses

$$v(x) = \begin{cases} x^\alpha & \text{if } x \geqslant 0 \\ -\lambda(-x)^\alpha & \text{if } x < 0 \end{cases}$$

for the value function and

$$w(p) = \frac{p^\delta}{(p^\delta + (1-p)^\delta)^{1/\delta}}$$

for the weighting function. Psychological studies suggest $\alpha = 0.88$, $\lambda = 2.25$, $\delta = 0.65$.

Under gaussian assumptions (more generally, in the absence of skewness), cumulative prospect theory is consistent with the CAPM (capital asset pricing model); however, the weighting function creates mispricing for assets with skewed returns – but the authors are not convinced that it can be arbitraged away.

Cumulative prospect theory can explain that assets with a high idiosyncratic skewness, such as IPOs, private equity, distressed stocks, deep out-of-the-money options, are overpriced and earn a low average return – this leads to the *volatility smile*. It also explains why household portfolios lack diversification.

To test (and use) that positively skewed stocks earn lower average returns, one would need to forecast future skewness: past skewness does not work, but cross-sectional industry skewness does.

Downside consumption risk and expected returns
V. Polkovnichenko (2006)

Yet another article showing that investors are averse to dowside risk: with *rank-dependent expected utility* (RDEU), utility is weighted by *decision weights* which are transformations of the cumulative objective probabilities of ranked events – *cumulative prospect theory* is a special case of RDEU.

The promise and peril of real options
A. Damodaran

Standard discounted cash flow models assume that the cash flows are deterministic; they are actually probabilistic and even contain *embedded options*, which have to be taken into account to properly value a cash flow or any corporate decision.

Examples include: the option to delay, expand, abandon a project, patents, natural ressources, etc.

The article recalls what an option is.

Evolution analysis of large-scale software systems using design structure matrix and design rule theory
M.J. LaMantia et al.

Modular software is good: the option (as in "option pricing" – these are *real options*) to replace a component makes it more valuable – a counter-example being the "complexity disaster" Windows Vista.

The *design structure matrix* (DSM) represents dependencies between the modules; once spotted, circular dependencies can be removed by adding one more module.