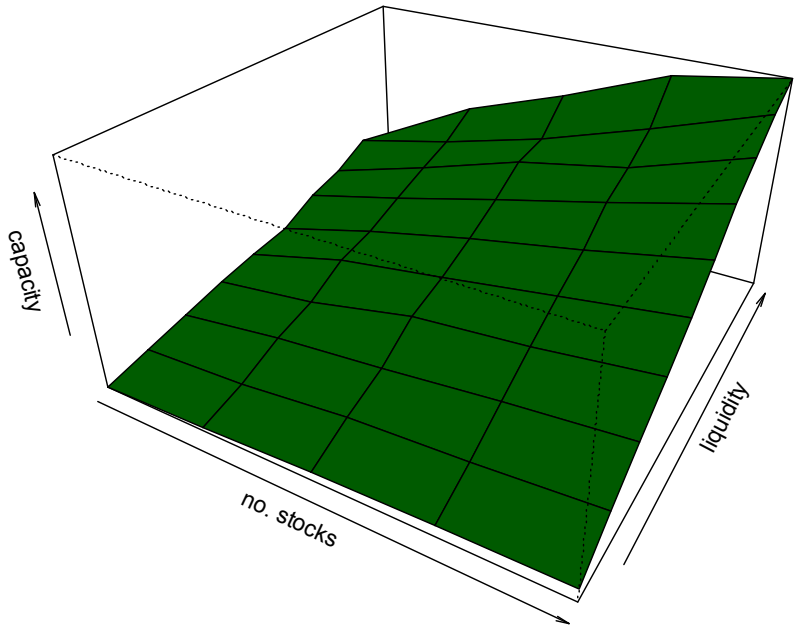
December 2011

Practical Optimization

Richard Saldanha
Investec Asset Management



Problem

- Estimate maximum monetary size of an equity portfolio using realistic assumptions and specific constraints
- Input data: stock prices, average or median daily volumes, market capitalization, stock weights (index-derived perhaps) at specific point in time
- Multiple portfolios may overlap with one another in terms of stock names – impacting estimates
- Not necessarily seeking optimality or perfect solution – more exploration of sample space

Problem restated

Maximize: $\boldsymbol{p}'\boldsymbol{x}$

subject to $A\boldsymbol{x} \leq \boldsymbol{b}$ and $\boldsymbol{x} \geq 0$

where

$\boldsymbol{p}$ is an $n \times 1$ real-valued vector of (fixed) prices ($\boldsymbol{p} \geq 0$)

$\boldsymbol{x}$ is an $n \times 1$ integer (but could be real-valued) vector of holdings or positions ($\boldsymbol{x} \geq 0$)

$\boldsymbol{b}$ is an $m \times 1$ real-valued vector of constraints

A is an $m \times n$ real-valued matrix such that $A\boldsymbol{x} \leq \boldsymbol{b}$

Solution

- Find positions ($\boldsymbol{x} \geq 0$) that maximize sum of prices $\times$ positions ($\boldsymbol{p}'\boldsymbol{x}$) subject to set of linear inequality constraints ($A\boldsymbol{x} \leq \boldsymbol{b}$)
- Classical linear programming problem or optimization (maximization) of linear objective function ($\boldsymbol{p}'\boldsymbol{x}$)

CRAN Task View: Optimization and Mathematical Programming

Classification According to Subject

- LP (Linear programming, 90C05): boot, glpk, limSolve, linprog, lpSolve, lpSolveAPI, rcdd, Rcplex, Rglpk, Rsymphony, quantreg
- GO (Global Optimization): Rdonlp2
- SPLP (Special problems of linear programming like transportation, multi-index, etc., 90C08): clue, lpSolve, lpSolveAPI, optmatch, quantreg, TSP
- BP (Boolean programming, 90C09): glpk, Rglpk, lpSolve, lpSolveAPI, Rcplex
- IP (Integer programming, 90C10): glpk, lpSolve, lpSolveAPI, Rcplex, Rglpk, Rsymphony
- MIP (Mixed integer programming and its variants MILP for LP and MIQP for QP, 90C11): glpk, lpSolve, lpSolveAPI, Rcplex, Rglpk, Rsymphony
- SP (Stochastic programming, 90C15): stoprog
- QP (Quadratic programming, 90C20): kernlab, limSolve, LowRankQP, quadprog, Rcplex
- SDP (Semidefinite programming, 90C22): Rcsdp
- MOP (Multi-objective and goal programming, 90C29): goalprog, mco
- NLP (Nonlinear programming, 90C30): Rdonlp2, Rsolnp
- GRAPH (Programming involving graphs or networks, 90C35): igraph, sna
- IPM (Interior-point methods, 90C51): kernlab, glpk, LowRankQP, quantreg, Rcplex
- RGA (Methods of reduced gradient type, 90C52): stats (optim()), gsl
- QN (Methods of quasi-Newton type, 90C53): stats (optim()), gsl, ucminf
- DF (Derivative-free methods, 90C56): minqa

Constraints

Five basic sets of constraints:

1. Each $p_i x_i$ must be less than some market capitalization threshold for each stock i in $1,...,n$
2. each $p_i x_i$ must be less than a predefined weight multiplied by the theoretical maximum size of the portfolio (shape constraint)
3. each $p_i x_i$ must be less than some non-negative constant liquidity factor, λ, multiplied by daily monetary volume of stock i, e.g. setting λ equal to 5 would imply 20 days to liquidate position i if 25% of daily volume is traded
4. each $p_i x_i$ must be greater than some minimum weight
5. each $p_i x_i$ must be less than some maximum weight

Setting up constraints $\boldsymbol{Ax} \leq \boldsymbol{b}$

- A is an $m \times n$ matrix (here $m = 5n$) constructed from a set of three $n \times n$ diagonal matrices, $\text{diag}(\boldsymbol{p})$, stacked below one another; plus two more $n \times n$ matrices, $\boldsymbol{lp}' - \text{diag}(\boldsymbol{p})$ and $\text{diag}(\boldsymbol{p}) - \boldsymbol{up}'$, stacked below the first three matrices, where $0 \leq \boldsymbol{l} < 1$ is an $n \times 1$ real-valued vector of minimum weights and $0 < \boldsymbol{u} \leq 1$ is an $n \times 1$ real-valued vector of maximum weights
  - note that $\text{diag}(\boldsymbol{p})\boldsymbol{x} \geq \boldsymbol{lp}'\boldsymbol{x}$ defines minimum (lower) weight constraints so that $\{\boldsymbol{lp}' - \text{diag}(\boldsymbol{p})\}\boldsymbol{x} \leq 0$; similarly, $\text{diag}(\boldsymbol{p})\boldsymbol{x} \leq \boldsymbol{up}'\boldsymbol{x}$ defines maximum (upper) weight constraints so that $\{\text{diag}(\boldsymbol{p}) - \boldsymbol{up}'\}\boldsymbol{x} \leq 0$
- The corresponding $\boldsymbol{b}$ is an $m \times 1$ vector comprising an $n \times 1$ vector defining maximum per stock market capitalization thresholds, an $n \times 1$ vector specifying a portfolio shape constraint, an $n \times 1$ vector specifying per stock liquidity constraints and a further $2n \times 1$ vector containing zeros (the right hand side of the minimum and maximum weight constraints) all stacked one upon the other
- Additional constraints (just linear combinations) are easily incorporated
- No general requirement for m to be greater than, equal to or less than n

Linear programming and simplex method

We seek

- Some *feasible vector* – $\boldsymbol{x}$ that satisfies the stated constraints $A\boldsymbol{x} \leq \boldsymbol{b}$
- *Optimal feasible vector* maximizes the *objective function* $\boldsymbol{p}'\boldsymbol{x}$
  - Can fail to exist for two reasons:
    1. no feasible vectors
    2. no maximum (unbounded space where one or more of the x_i belonging to $\boldsymbol{x}$ can be taken to infinity without violating constraints) $\boldsymbol{p}'\boldsymbol{x}$ is then unbounded

Method

- Start with full n-dimensional space of candidate vectors
- Carve away regions eliminated by each imposed constraint
- We are left with some *feasible region* or else there are no feasible vectors

- Since the feasible region is bounded by hyperplanes, geometrically it is a *simplex* (e.g. a generalised triangle in two dimensions or tetrahedron in three-dimensions)
- Feasible vectors lie on the boundary of the feasible region not in the region's interior
  - we just have to find which feasible vector is best
  - optimal feasible vector (if it exists) lies at a vertex (corner) of simplex
- Danzig's 1948 Simplex Method[1] organizes the search so that:
  1. the objective function increases in value at each step (i.e. algorithm always heads in correct direction)
  2. the optimal feasible vector is reached after number of iterations almost no larger than order m or n, whichever is larger (i.e. algorithm is efficient)
- All this is easily set up using Open Source *R* (www.r-project.org) and the `linprog` package from Arne Henningsen; see Appendix

[1]Don't confuse Danzig's method with the Nelder–Mead simplex method for minimizing an objective function. Nelder–Mead is a general nonlinear optimization technique that uses a varying simplex to conduct its search for a minimum of an objective function.

Exploring the sample space

- Look at randomly chosen portfolios of size n from stock universe
- Logical to sample w.r.t. to some $\boldsymbol{w}$, e.g. set of benchmark or universe stock weights, which could simply be vector of equal weights each $1/n$

R in action

Free-float market cap

Free-float weight

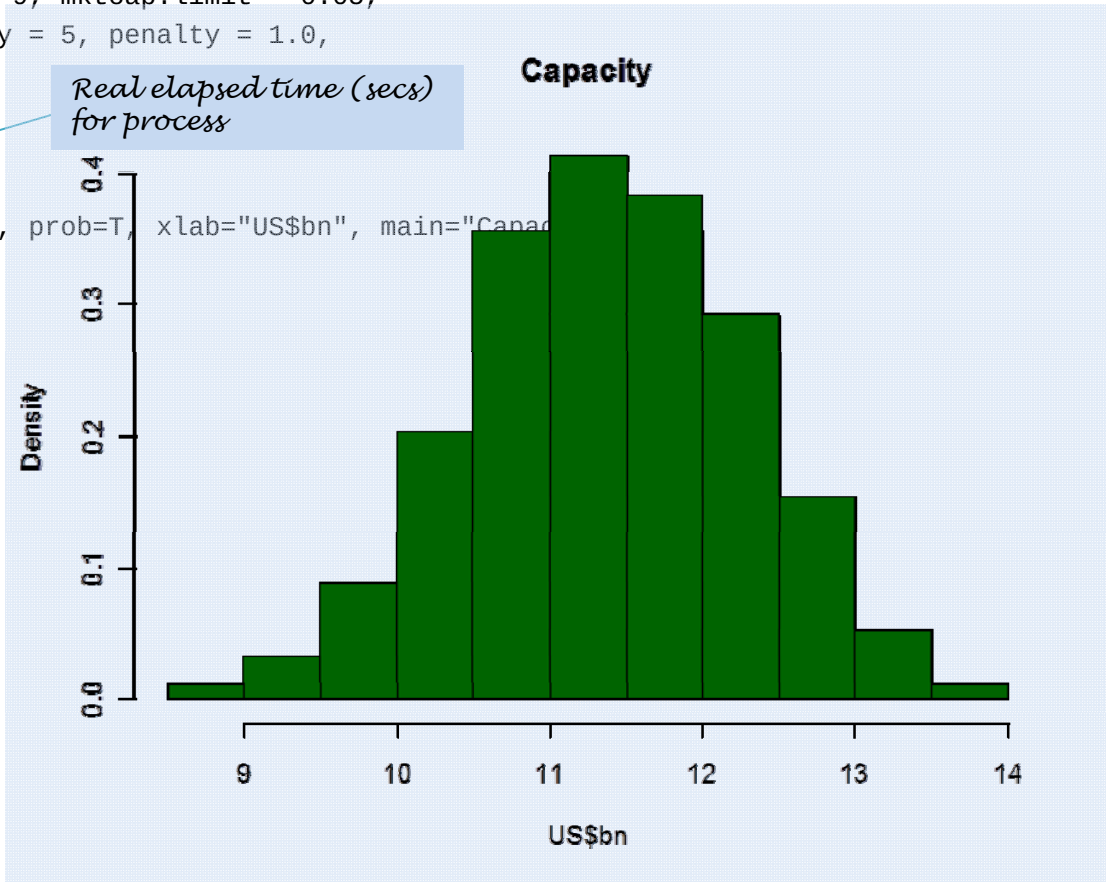
```
> head(X)
    bloomberg                 name cc ccy usdprice mktcap.usdbn   weight ffmktcap.usdbn ffweight    volume
  1    857 HK  PETROCHINA CO LTD-H CN HKD      1.4        303.2 0.043267          301.1 0.071064 101232800
  2   1398 HK IND & COMM BK OF CHI CN HKD      0.8        248.0 0.035385          201.5 0.047571 256751700
  3   3988 HK  BANK OF CHINA LTD-H CN HKD      0.5        144.8 0.020664          107.7 0.025411 270147400
  4    386 HK CHINA PETROLEUM & CH CN HKD      1.0        102.6 0.014643          102.6 0.024221 102057000
  5 005930 KS SAMSUNG ELECTRONICS  KR KRW    822.5        121.2 0.017287           91.0 0.021479    444708
  6   2628 HK CHINA LIFE INSURANCE CN HKD      3.4         86.3 0.012316           86.3 0.020374  42403400
> tail(X)
    bloomberg                 name cc ccy usdprice mktcap.usdbn   weight ffmktcap.usdbn ffweight    volume
600 003470 KS TONG YANG SECURITIES KR KRW      5.8          0.7 0.000104            0.4  9.9e-05    642576
601   3227 TT   PIXART IMAGING INC TW TWD      3.6          0.5 0.000067            0.4  9.4e-05   1571711
602   2315 TT  MITAC INTERNATIONAL TW TWD      0.4          0.6 0.000081            0.4  9.2e-05   5659210
603   2607 TT EVERGREEN INTERNATIO TW TWD      0.8          0.9 0.000128            0.4  8.6e-05   4170820
604   5347 TT VANGUARD INTERNATION TW TWD      0.5          0.9 0.000128            0.4  8.5e-05   5069453
605    MAS MK MALAYSIAN AIRLINE SY MY MYR      0.5          1.6 0.000228            0.3  7.4e-05   1809906
>
```

3-month ADV

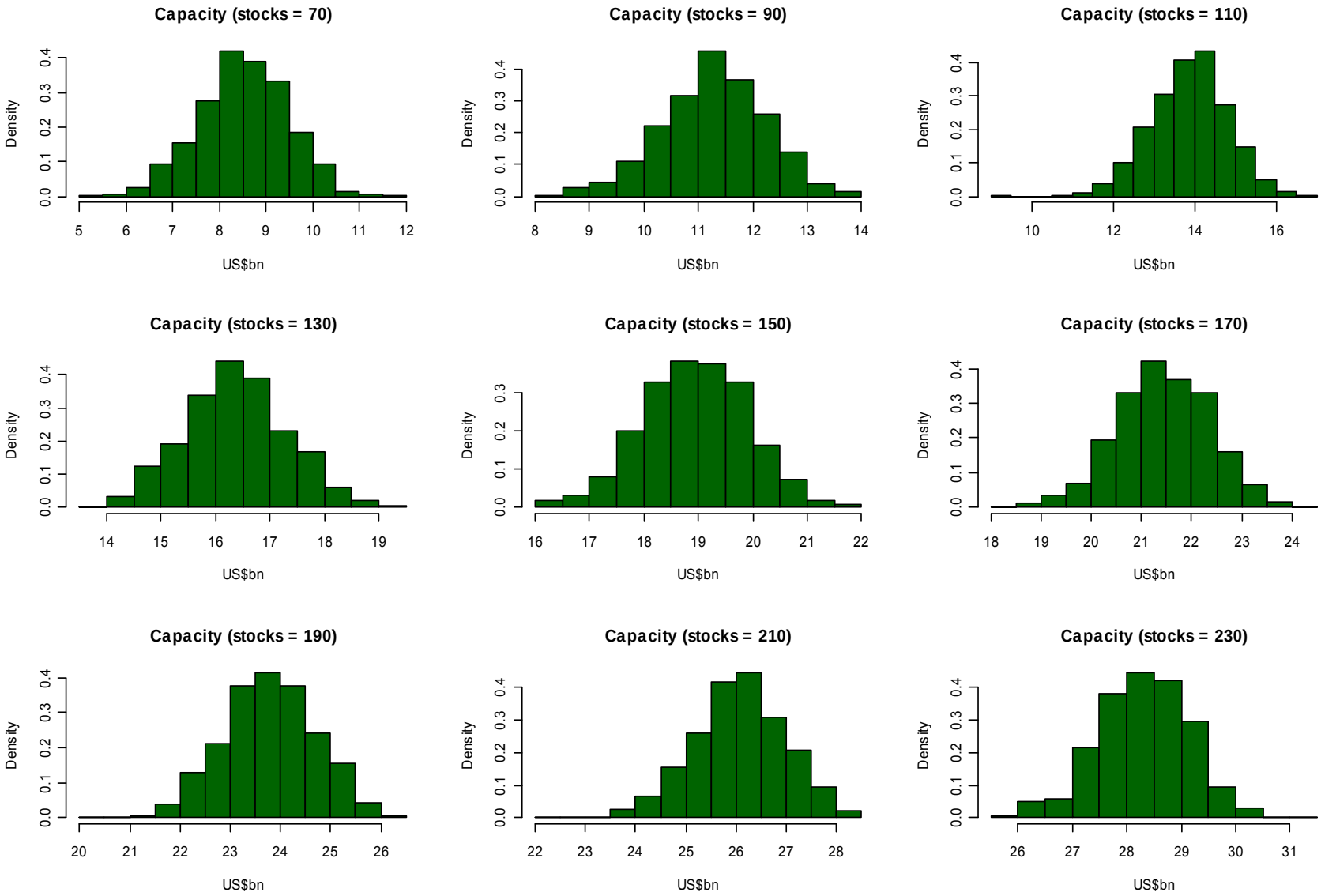
Units matter - capacity function expects common price and it's easiest to supply market cap to function in base units i.e. multiply mktcap.usdbn by 10^9 on entry

```
> library(linprog)
Loading required package: lpSolve
> dim(X)
[1] 605  6
> system.time(
+ X.results <- capacity.sample(nsample = 1000, nstocks = 91,
+ name = X[,"bloomberg"], price = X[,"usdprice"], volume = X[,"volume"],
+ mktcap = X[,"ffmktcap.usdbn"] * 10^9, mktcap.limit = 0.05,
+ weight = X[, "ffweight"], liquidity = 5, penalty = 1.0,
+ lpSolve = T, maxiter = 500)
+ )
   user  system elapsed
   8.05    0.95   10.84
> hist(X.results$portfolio.size/10^9, prob=T, xlab="US$bn", main="Capac
```

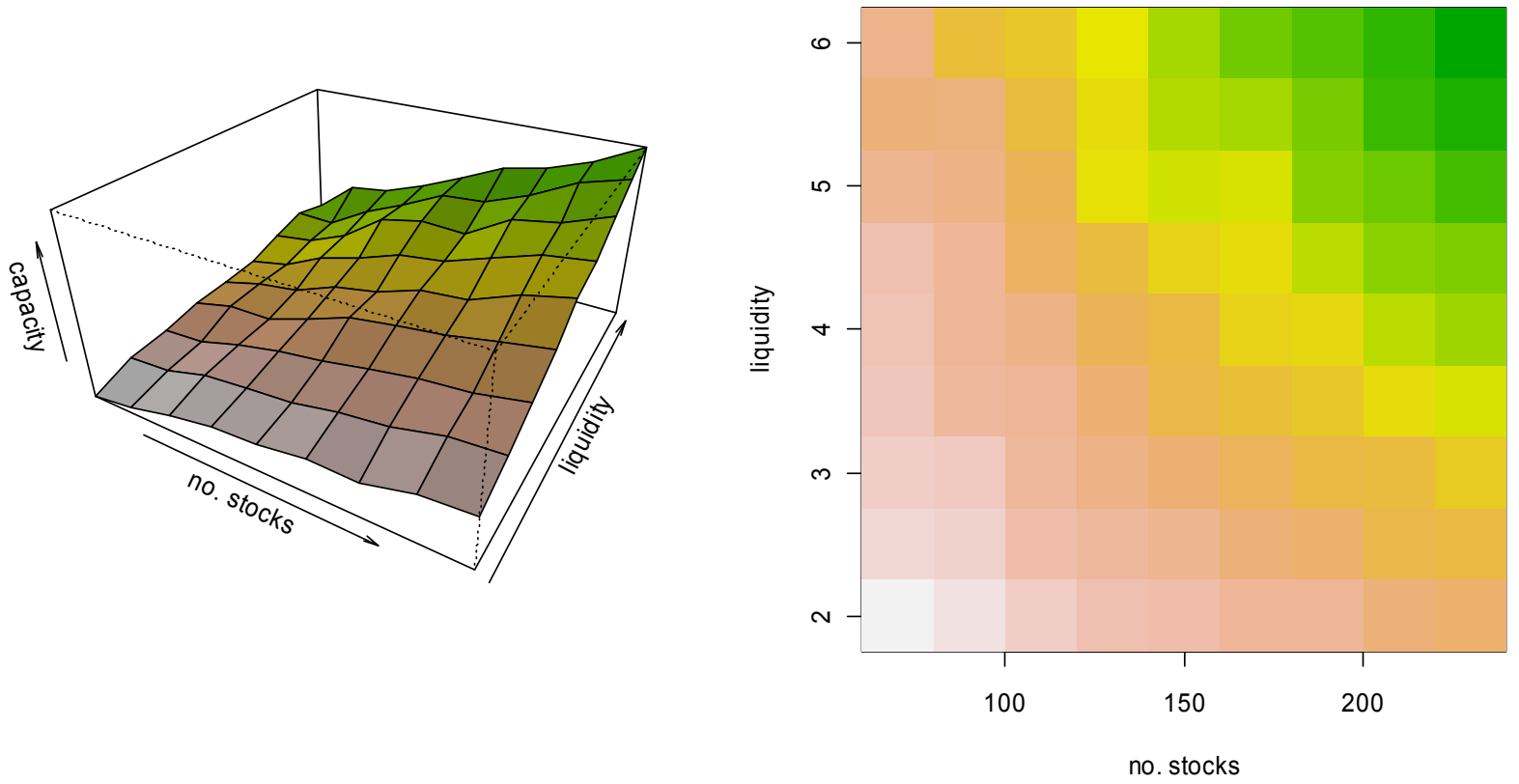
Real elapsed time (secs) for process



Distribution of the maximal capacity of 1000 randomly selected portfolios containing 91 out of 605 stocks



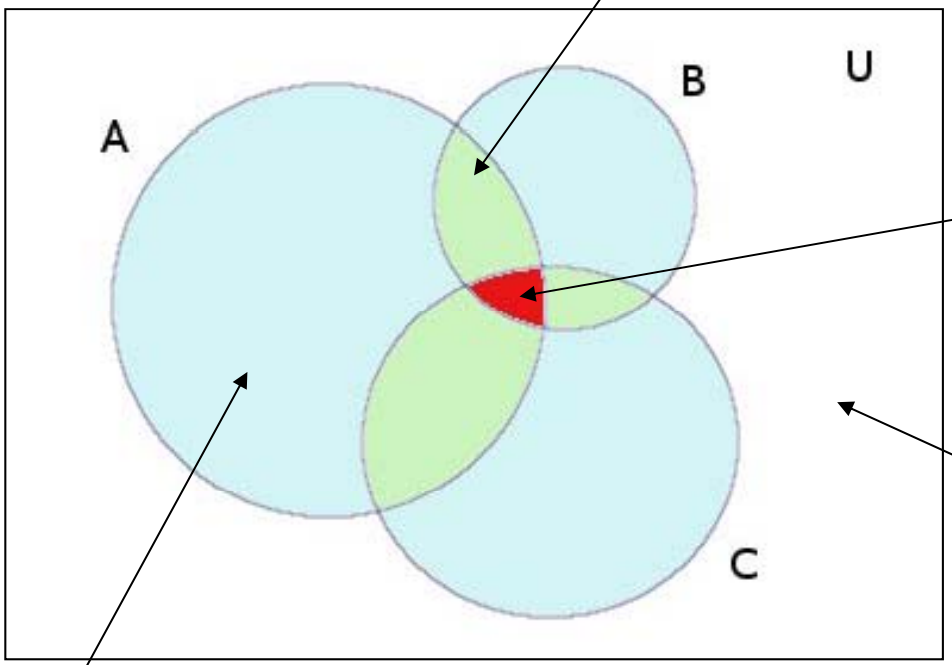
Capacity distributions as the number of stocks in a portfolio increases. (Each histograms shows the results of 1000 simulations.)



Perspective and image plots showing how maximum capacity varies by the number of stocks in a portfolio and liquidity of those stocks.

Modelling overlapping portfolios

U is a Universe of stocks
A, B & C are subsets of U



Green shaded regions have three overlaps A ∩ B ∩ U

Red region has four overlaps A ∩ B ∩ C ∩ U

Region with no overlap

Blue shaded regions have two overlaps, i.e. A ∩ U or B ∩ U or C ∩ U

Penalty function

- Let $\boldsymbol{\eta}$ be an $n \times 1$ integer vector counting the number of stock groups in which each stock appears
- Define a penalty function over $(0, 1]$

$$g(\boldsymbol{\eta}) = 1 / \{1 + a\log(\boldsymbol{\eta})\}, \qquad (1)$$

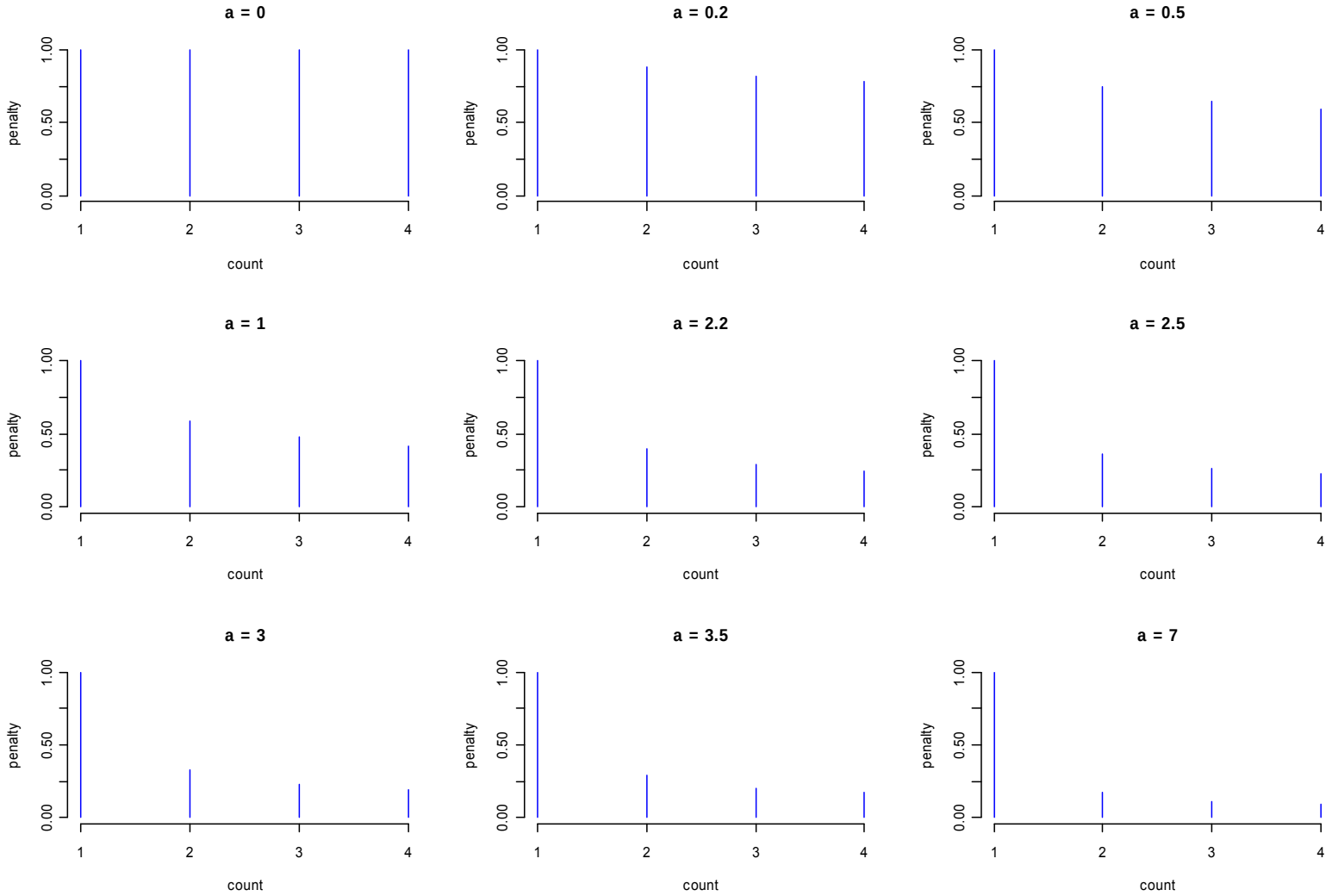
where a is some suitably chosen real-valued constant such that the higher the count (more stock-group overlap) the smaller g is

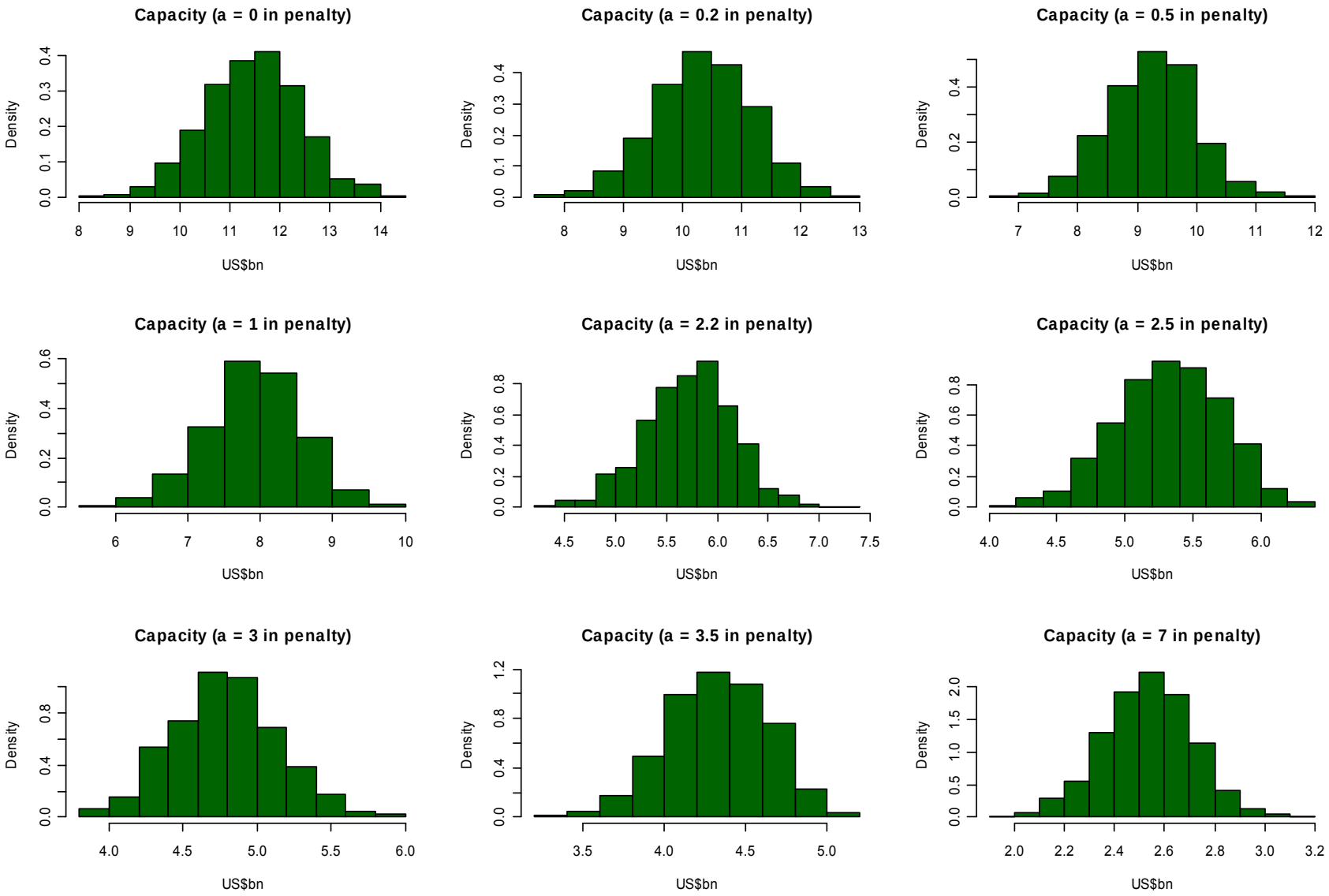
- Use this to penalize the liquidity of stocks that appear in more than one group
- Set $a = (1 - g^{\dagger}) / g^{\dagger} \log(\eta)$ for specific weight $g^{\dagger} = g(\eta)$ and integer count, η, to 'tune' penalty, e.g.

  setting $g^{\dagger} = ¼$ for $\eta = 4$ implies $a \approx 2.16$ in (1)

  setting $g^{\dagger} = ¾$ for $\eta = 2$ implies $a \approx 0.48$ in (1)

Penalty for selected values of a





Capacity distributions for four overlapping groups of stocks as the degree of penalty for overlaps is increased. (Each histogram shows the results of 1000 simulations.)

Final comments

- Seemingly stable and coherent framework
- Ability to extend
- Use of penalty allows one-at-a-time experimentation rather than complex joint modelling
- As with any stochastic model it is an imperfect representation of real world
- Snapshot in time of prices, volumes, etc. might be fine today but poor tomorrow
- Doesn’t consider market impact in terms of price nor degradation of returns as portfolios get larger


Alex
PEATTIE + TAYLOR
I'VE GOT A PHD IN NUCLEAR PHYSICS. I NEVER IMAGINED I'D END UP WORKING IN A BANK...
alex@alexcartoon.com
I THOUGHT I'D BE DISCOVERING NEW SUBATOMIC PARTICLES OR POSTULATING THEORIES ABOUT THE ORIGINS OF THE UNIVERSE...
YOU MAY NOT BE SERVING HUMANITY AT LARGE BUT YOU'RE INVALUABLE TO US...
HAVING QUANTS LIKE YOU WITH THE MATHEMATICAL SKILLS TO CONSTRUCT COMPLEX ALGORITHMS CAPABLE OF HANDLING MULTIPLE PARAMETERS AND VARIABLES IS VITAL TO THE BANK...
YES, BUT DOING THE SEATING PLAN FOR THE NEW OFFICE MOVE?
HERE'S A FULL LIST OF ALL THE PEOPLE WHO REFUSE TO SIT NEXT TO EACH OTHER... AND ALL THOSE WHO ARE DEMANDING CORNER OFFICES...

Appendix: `solveLP`

Solve Linear Programming / Optimization Problems

Description

Minimizes (or maximizes) c'x, subject to A x <= b and x >= 0.

Note that the inequality signs <= of the individual linear constraints in A x <= b can be changed with argument const.dir.

Usage

```
solveLP( cvec, bvec, Amat, maximum = FALSE,
   const.dir = rep( "<=", length( bvec ) ),
   maxiter = 1000, zero = 1e-9, tol = 1e-6, dualtol = tol,
   lpSolve = FALSE, solve.dual = FALSE, verbose = 0 )
```

Arguments

cvec	vector c (containing n elements).
bvec	vector b (containing m elements).
Amat	matrix A (of dimension m \times n).
maximum	logical. Should we maximize or minimize (the default)?
const.dir	vector of character strings giving the directions of the constraints: each value should be one of "<," "<=," "=," "==," ">," or ">=". (In each pair the two values are identical.)

maxiter	maximum number of iterations.
zero	numbers smaller than this value (in absolute terms) are set to zero.
tol	if the constraints are violated by more than this number, the returned component status is set to 3.
dualtol	if the constraints in the dual problem are violated by more than this number, the returned status is non-zero.
lpSolve	logical. Should the package 'lpSolve' be used to solve the LP problem?
solve.dual	logical value indicating if the dual problem should also be solved.
verbose	an optional integer variable to indicate how many intermediate results should be printed (0 = no output; 4 = maximum output).

Details

This function uses the Simplex algorithm of George B. Dantzig (1947) and provides detailed results (e.g. dual prices, sensitivity analysis and stability analysis). If the solution x=0 is not feasible, a 2-phase procedure is applied. Values of the simplex tableau that are actually zero might get small (positive or negative) numbers due to rounding errors, which might lead to artificial restrictions. Therefore, all values that are smaller (in absolute terms) than the value of zero (default is 1e-10) are set to 0.

Solving the Linear Programming problem by the package lpSolve (of course) requires the installation of this package, which is available on CRAN (http://cran.r-project.org/src/contrib/PACKAGES.html#lpSolve). Since the lpSolve package uses C-code and this (linprog) package is not optimized for speed, the former is much faster. However, this package provides more detailed results (e.g. dual values, stability and sensitivity analysis).

This function has not been tested extensively and might not solve all feasible problems (or might even lead to wrong results). However, you can export your LP to a standard MPS file via writeMps and check it with other software (e.g. lp_solve, see ftp://ftp.es.ele.tue.nl/pub/lp_solve).

Equality constraints are not implemented yet.

Value

solveLP returns a list of the class solveLP containing following objects:

opt	optimal value (minimum or maximum) of the objective function.
solution	vector of optimal values of the variables.
iter1	iterations of Simplex algorithm in phase 1.
iter2	iterations of Simplex algorithm in phase 2.
basvar	vector of basic (=non-zero) variables (at optimum).
con	matrix of results regarding the constraints:
1st column	= maximum values (=vector b);
2nd column	= actual values;
3rd column	= differences between maximum and actual values;
4th column	= dual prices (shadow prices);
5th column	= valid region for dual prices.

allvar matrix of results regarding all variables (including slack variables):
1st column = optimal values;
2nd column = values of vector c;
3rd column = minimum of vector c that does not change the solution;
4th column = maximum of vector c that does not change the solution;
5th column = derivatives to the objective function;
6th column = valid region for these derivatives.
status numeric. Indicates if the optimization did succeed:
0 = success;
1 = lpSolve did not succeed;
2 = solving the dual problem did not succeed;
3 = constraints are violated at the solution (internal error or large rounding errors);
4 = simplex algorithm phase 1 did not find a solution within the number of iterations specified by argument maxiter;
5 = simplex algorithm phase 2 did not find the optimal solution within the number of iterations specified by argument maxiter.
lpStatus numeric. Return code of lp (only if argument lpSolve is TRUE).
dualStatus numeric. Return code from solving the dual problem (only if argument solve.dual is TRUE).
maximum logical. Indicates whether the objective function was maximized or minimized.
Tab final 'Tableau' of the Simplex algorith.
lpSolve logical. Has the package 'lpSolve' been used to solve the LP problem.
solve.dual logical. Argument solve.dual.
maxiter numeric. Argument maxiter.

Author(s)

Arne Henningsen

References

Dantzig, George B. (1951) Maximization of a linear function of variables subject to linear inequalities, in Koopmans, T.C. (ed.), *Activity analysis of production and allocation*. John Wiley \& Sons, New York, p. 339-347.

Steinhauser, Hugo; Cay Langbehn and Uwe Peters (1992) Einfuehrung in die landwirtschaftliche Betriebslehre. Allgemeiner Teil, 5th ed., Ulmer, Stuttgart.

Witte, Thomas; Joerg-Frieder Deppe and Axel Born (1975) *Lineare Programmierung*. Einfuehrung fuer Wirtschaftswissenschaftler, Gabler-Verlag, Wiesbaden.