Efficiently computing annuity conversion factors via feed-forward neural networks

Insurance Data Science Conference, Hanover, 10.06.2026

Sascha Günther, ETH Zürich

(joint work with Peter Hieber (University of Lausanne)

and Maria Aragona (University Torino))

ETH zürich

The Setting

Setting

We have an economic scenario generator with risk factors $\mathcal{Z}$:

- interest rate risk
- longevity risk
- inflation
- stock market,
- ...

An insurance company guarantees an **annuity conversion factor** at retirement (in $T \in \mathbb{N}$ years from now)

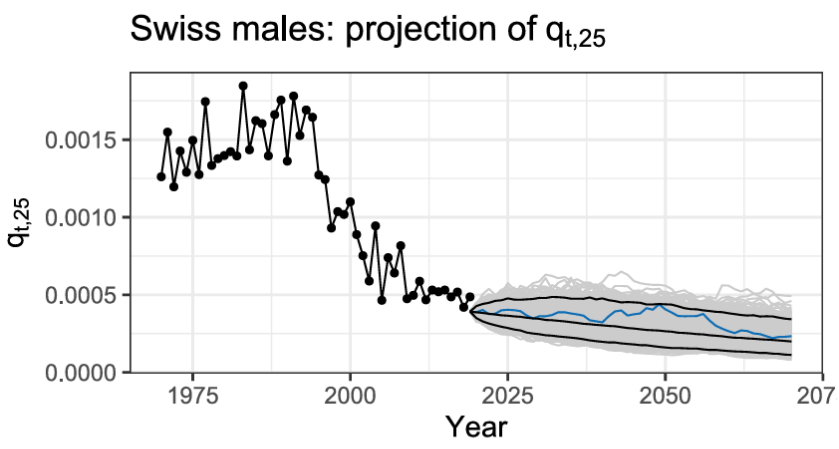
Setting

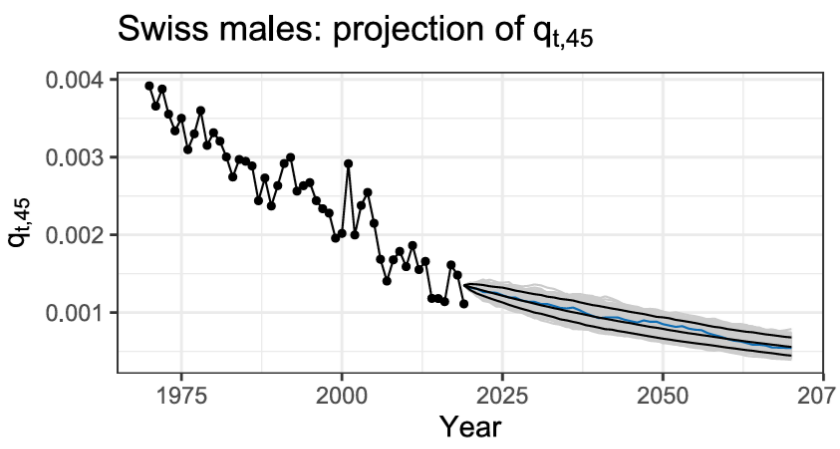
We have an economic scenario generator with risk factors Z:

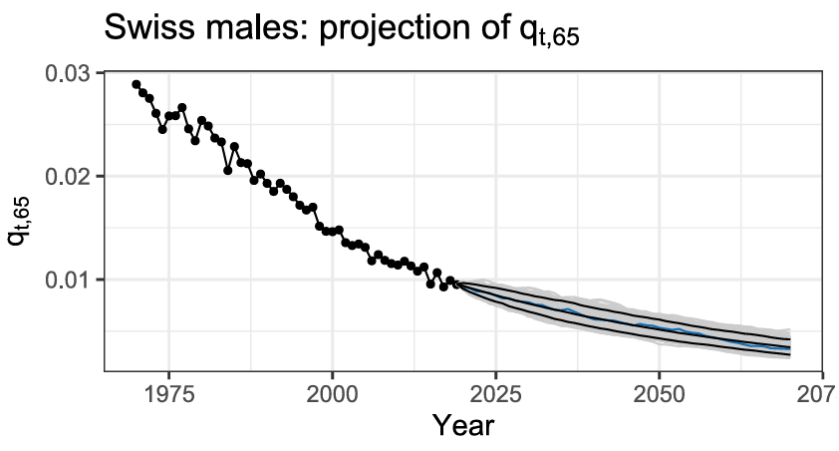
- interest rate risk
- longevity risk
- inflation
- stock market,
- ...

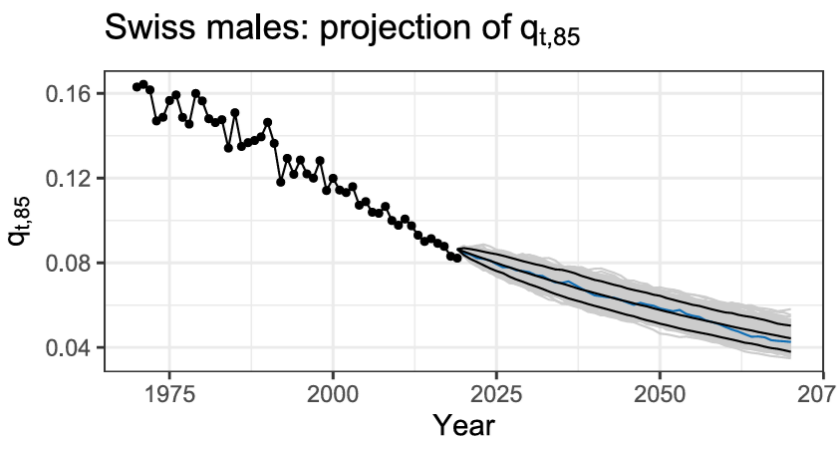
An insurance company guarantees an **annuity conversion factor** at retirement (in $T \in \mathbb{N}$ years from now)

Mortality risk









ETH zürich

Example: 4 risk factors

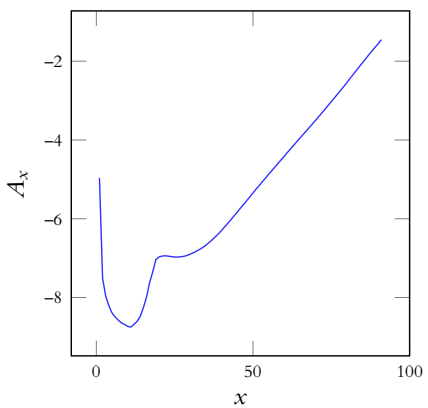
- For **longevity risk**, use **2-factor** Li-Lee model (see Li and Lee (2005)) with factors k_t and K_T

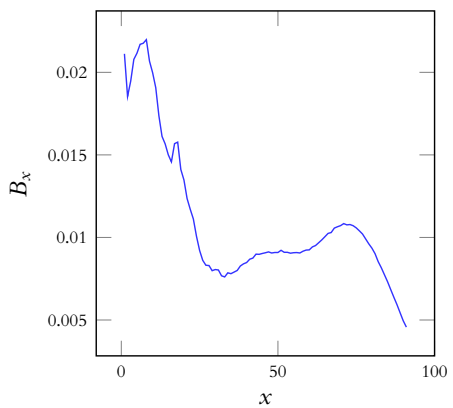
- For **interest rate risk** use **2-factor** Hull-White / G2++ model (see also Graf and Korn (2020), Günther and Hieber (2024))

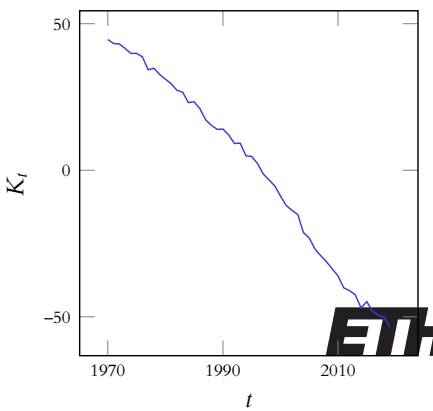
Example: 4 risk factors

- For **longevity risk**, use **2-factor** Li-Lee model (see Li and Lee (2005)) with factors k_t and K_T

$$\ln(m_{x,t}) = \ln(m^{\mathrm{g}}_{x,t}) + \ln(m^{\mathrm{c}}_{x,t})\,,$$
$$\ln(m^{\mathrm{g}}_{x,t}) = A_x + B_x K_t\,,$$
$$\ln(m^{\mathrm{c}}_{x,t}) = a_x + b_x k_t\,,$$
$$K_t = K_{t-1} + \gamma + \epsilon_t\,,$$
$$k_t = c + d \cdot k_{t-1} + \delta_t\,.$$







Example: 4 risk factors

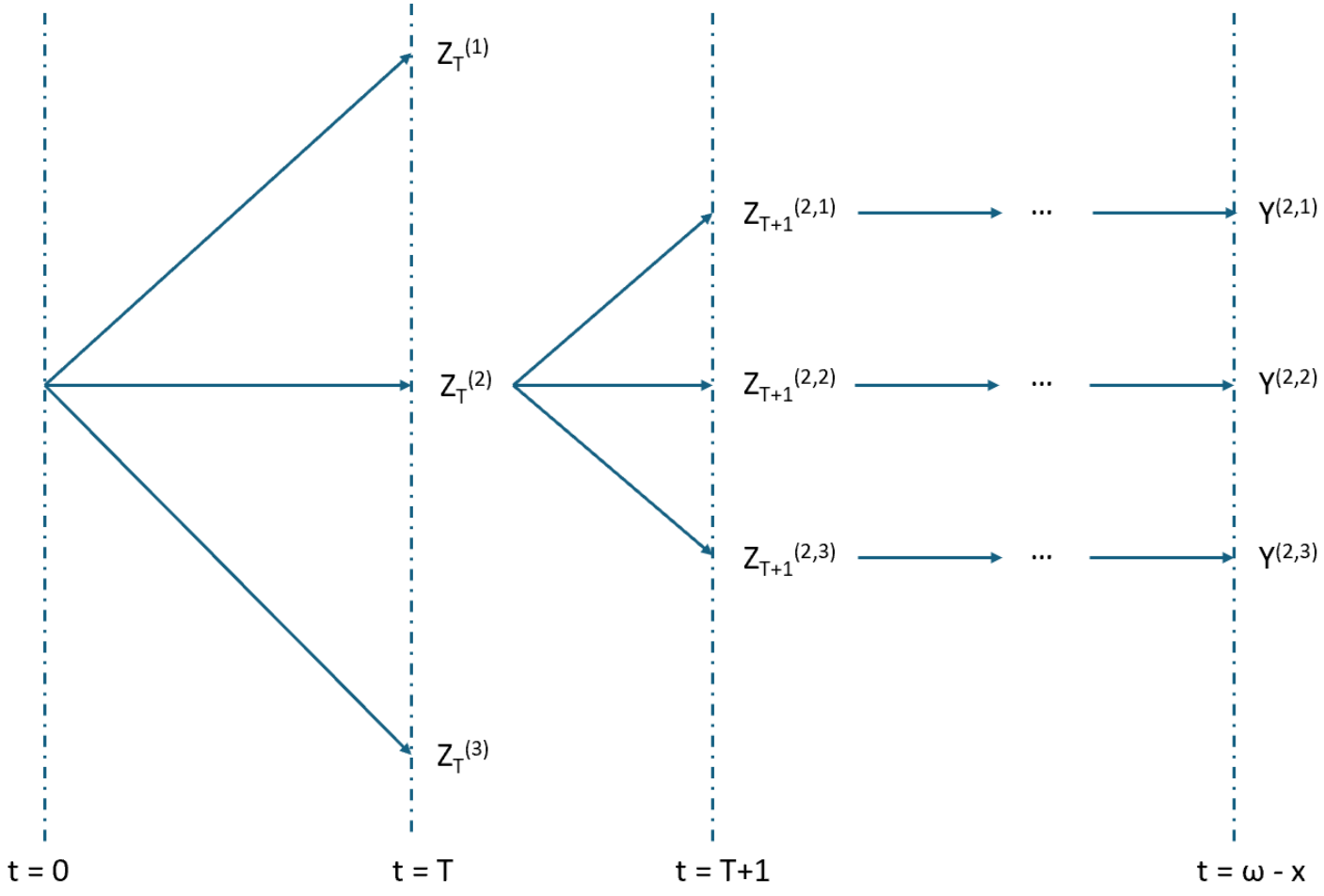
- For **longevity risk**, use **2-factor** Li-Lee model (see Li and Lee (2005)) with factors k_t and K_T

- For **interest rate risk** use **2-factor** Hull-White / G2++ model (see also Graf and Korn (2020), Günther and Hieber (2024))

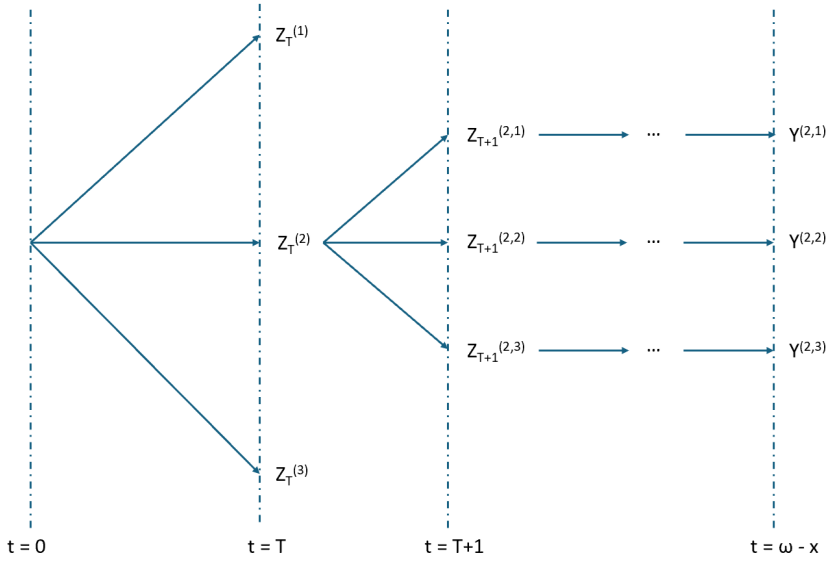
$$
\begin{aligned}
r_t &= \psi_t + x_t + y_t\,, \\
\mathrm{d}x_t &= -a\,x_t\,\mathrm{d}t + \nu\,\mathrm{d}W_t^{(1)}\,, \\
\mathrm{d}y_t &= -b\,y_t\,\mathrm{d}t + \eta\Big(\rho_r\,\mathrm{d}W_t^{(1)} + \sqrt{1-\rho_r^2}\,\mathrm{d}W_t^{(2)}\Big)\,, \\
\psi_t &= f^M(0,t) + \text{volatility correction}\,,
\end{aligned}
$$

where $f^M(0,t)$ is a Nelson-Siegel-Svensson yield curve.

Nested simulations



Nested simulations



$$\begin{aligned}\text{Number of simulation steps} &= \text{outer scenarios} \times \text{inner scenarios} \times \text{time steps} \\ &= 1\,000\,000 \times 1\,000 \times 60 = 6 \cdot 10^{10} \\ \text{time taken} &= \text{Number of steps} \times \text{time taken per step} \\ &= 6 \cdot 10^{10} \cdot 1\,\text{ms} \approx 1.9 \text{ years}\end{aligned}$$

ETH zürich

Inner estimation

- Risk factors are stored in the (random) vector $\{\boldsymbol{Z}_t\}_{t\geq 0}$
  e.g. $\boldsymbol{Z}_t = (K_t, k_t, x_t, y_t)$

- We need to **efficiently evaluate** the annuity conversion factor **conditional on** $\boldsymbol{Z}_T = \boldsymbol{z}$:

$$a_x(T, \boldsymbol{z}) = \mathbb{E}\left[\sum_{t=1}^{\omega-x-T} \mathrm{e}^{-\int_T^{T+t} r_s \,\mathrm{d}s} \mathrm{e}^{-\sum_{s=0}^{t-1} m_{x+T+s,T+s}} \,\middle|\, \boldsymbol{Z}_T = \boldsymbol{z} \right].$$

There is no closed-form solution for this expectation!

- The output to predict is thus $\mathbb{E}[Y \mid \boldsymbol{Z}_T = \boldsymbol{z}] = a_x(T, \boldsymbol{z})$

Inner estimation

- Risk factors are stored in the (random) vector $\{\boldsymbol{Z}_t\}_{t\geq 0}$
  e.g. $\boldsymbol{Z}_t = (K_t, k_t, x_t, y_t)$

- We need to **efficiently evaluate** the annuity conversion factor **conditional on** $\boldsymbol{Z}_T = \boldsymbol{z}$:

$$a_x(T, \boldsymbol{z}) = \mathbb{E}\left[\sum_{t=1}^{\omega-x-T} \mathrm{e}^{-\int_T^{T+t} r_s \,\mathrm{d}s} \mathrm{e}^{-\sum_{s=0}^{t-1} m_{x+T+s,T+s}} \,\middle|\, \boldsymbol{Z}_T = \boldsymbol{z} \right].$$

There is no closed-form solution for this expectation!

- The output to predict is thus $\mathbb{E}[Y \mid \boldsymbol{Z}_T = \boldsymbol{z}] = a_x(T, \boldsymbol{z})$

Inner estimation

- Risk factors are stored in the (random) vector $\{\boldsymbol{Z}_t\}_{t\geq 0}$
  e.g. $\boldsymbol{Z}_t = (K_t, k_t, x_t, y_t)$

- We need to **efficiently evaluate** the annuity conversion factor **conditional on** $\boldsymbol{Z}_T = \boldsymbol{z}$:

$$a_x(T, \boldsymbol{z}) = \mathbb{E}\left[\sum_{t=1}^{\omega-x-T} \mathrm{e}^{-\int_T^{T+t} r_s \,\mathrm{d}s} \mathrm{e}^{-\sum_{s=0}^{t-1} m_{x+T+s,T+s}} \,\middle|\, \boldsymbol{Z}_T = \boldsymbol{z} \right].$$

  There is no closed-form solution for this expectation!

- The output to predict is thus $\mathbb{E}[Y \mid \boldsymbol{Z}_T = \boldsymbol{z}] = a_x(T, \boldsymbol{z})$

Literature review

Literature (selection)

- **linear least-square regression** of inner simulations based on underlying risk factors:
  Longstaff and Schwartz (2001), Bauer et al., 2010, Floryszczak et al., 2016, Ha and Bauer, 2022, Bacinello et al., 2024

- **neural network** regressions:
  Gan (2013), Hejazi and Jackson (2017), Cheridito et al. (2020), Fernandez-Arjona and Filipović (2022), Barigou and Delong (2022)

- approaches using **Taylor expansions**:
  Biffis and Millossovich (2006), Dowd et al. (2011), Cairns (2011)

Literature (selection)

The articles closest to our research are:

- **Large datsets with many risk factors:** Castellani et al. (2021): internal model with approx. 20 factors,
  Krah et al. (2020), Jonen et al. (2023): 12 - 13 risk factors.
  Comparison of different techniques. Neural networks (should) show their strength.
- **Smaller models in actuarial science:** Bacinello et al. (2021) apply least-squares Monte-Carlo to annuity conversion factors.
  Fernandez-Arjona (2021) use a 4-factor model.

Surprising result: Neural networks have difficulties beating a simple polynomial least-square Monte-Carlo regression.

Literature (selection)

The articles closest to our research are:

- **Large datsets with many risk factors:** Castellani et al. (2021): internal model with approx. 20 factors,
  Krah et al. (2020), Jonen et al. (2023): 12 - 13 risk factors.
  Comparison of different techniques. Neural networks (should) show their strength.
- **Smaller models in actuarial science:** Bacinello et al. (2021) apply least-squares Monte-Carlo to annuity conversion factors.
  Fernandez-Arjona (2021) use a 4-factor model.

Surprising result: Neural networks have difficulties beating a simple polynomial least-square Monte-Carlo regression.

Literature (selection)

The articles closest to our research are:

- **Large datsets with many risk factors:** Castellani et al. (2021): internal model with approx. 20 factors,
  Krah et al. (2020), Jonen et al. (2023): 12 - 13 risk factors.
  Comparison of different techniques. Neural networks (should) show their strength.
- **Smaller models in actuarial science:** Bacinello et al. (2021) apply least-squares Monte-Carlo to annuity conversion factors.
  Fernandez-Arjona (2021) use a 4-factor model.

Surprising result: Neural networks have difficulties beating a simple polynomial least-square Monte-Carlo regression.

Our approach

Our approach

- "Interpolate" between feed-forward neural networks and least-squares Monte-Carlo:
  - use the (unusual) **polynomial activation** function $\varphi(x) = x^2$.
  - work with **small networks** (1 layer, $\approx$ 4-20 neurons)
  - Starting point: least-squares Monte-Carlo as **special case** of neural networks

Least-squares Monte-Carlo (LSMC)

- We choose:

$$a_x(T, \boldsymbol{z}) \approx e(\boldsymbol{z}) = \sum_{i=1}^{M} \beta_i e_i(\boldsymbol{z}) \,,$$

  where $e_i(\boldsymbol{z})$ are polynomial basis functions up to degree 2, e.g. $e_i(\boldsymbol{z}) = z_1^2$ or $e_i(\boldsymbol{z}) = z_2 \cdot z_3$.

- The weights β_i are determined according to a least-squares regression:

$$\hat{\boldsymbol{\beta}} = (\boldsymbol{X}'\boldsymbol{X})^{-1}\boldsymbol{X}'\boldsymbol{Y} \,.$$

Least-squares Monte-Carlo (LSMC)

- We choose:

$$a_x(T, \boldsymbol{z}) \approx e(\boldsymbol{z}) = \sum_{i=1}^{M} \beta_i e_i(\boldsymbol{z}) \,,$$

  where $e_i(\boldsymbol{z})$ are polynomial basis functions up to degree 2, e.g. $e_i(\boldsymbol{z}) = z_1^2$ or $e_i(\boldsymbol{z}) = z_2 \cdot z_3$.

- The weights β_i are determined according to a least-squares regression:

$$\hat{\boldsymbol{\beta}} = (\boldsymbol{X'X})^{-1}\boldsymbol{X'Y} \,.$$

Feed-Forward Neural Networks (NN)

We choose a **single-layer** neural network with activation functions φ_k, $k = 1, 2, \ldots, K$ in the hidden layer. The **final output** is thus

$$e(\boldsymbol{z}) = \gamma_0 + \sum_{k=1}^{K} \gamma_k \cdot \varphi_k \left(\alpha_{0,k} + \sum_{i=1}^{p} \alpha_{i,k} \cdot z_i \right) ,$$

where $\alpha_{i,k}$ is the i-th weight in the k-th neuron.

First choice: Exponential linear unit

$$\varphi_k(x) = \mathrm{Elu}(x) = \begin{cases} x, & \text{if } x \geq 0, \\ a(\mathrm{e}^x - 1), & \text{if } x < 0\,. \end{cases}$$

Second choice: Squared activation function

$$\varphi_k(x) = x^2\,.$$

ETH zürich

Feed-Forward Neural Networks (NN)

We choose a **single-layer** neural network with activation functions φ_k, $k = 1, 2, \ldots, K$ in the hidden layer. The **final output** is thus

$$e(\boldsymbol{z}) = \gamma_0 + \sum_{k=1}^{K} \gamma_k \cdot \varphi_k \left(\alpha_{0,k} + \sum_{i=1}^{p} \alpha_{i,k} \cdot z_i \right) ,$$

where $\alpha_{i,k}$ is the i-th weight in the k-th neuron.

First choice: Exponential linear unit

$$\varphi_k(x) = \mathrm{Elu}(x) = \begin{cases} x, & \text{if } x \geq 0, \\ a(\mathrm{e}^x - 1), & \text{if } x < 0\,. \end{cases}$$

Second choice: Squared activation function

$$\varphi_k(x) = x^2\,.$$

ETH zürich

Feed-Forward Neural Networks (NN)

We choose a **single-layer** neural network with activation functions φ_k, $k = 1, 2, \ldots, K$ in the hidden layer. The **final output** is thus

$$e(\boldsymbol{z}) = \gamma_0 + \sum_{k=1}^{K} \gamma_k \cdot \varphi_k \left(\alpha_{0,k} + \sum_{i=1}^{p} \alpha_{i,k} \cdot z_i \right) ,$$

where $\alpha_{i,k}$ is the i-th weight in the k-th neuron.

First choice: Exponential linear unit

$$\varphi_k(x) = \mathrm{Elu}(x) = \begin{cases} x, & \text{if } x \geq 0, \\ a(\mathrm{e}^x - 1), & \text{if } x < 0\,. \end{cases}$$

Second choice: Squared activation function

$$\varphi_k(x) = x^2\,.$$

ETH zürich

Numerical Results

Error measurement

- We determine the **90th percentile** $q_{90\%}^{\text{LSMC/NN}}$ of the annuity values for NN and LSMC.
- We compare the two procedures to the *true value* considering the **mean absolute percentage error** (MAPE):

$$\text{MAPE}^{\text{LSMC/NN}} = \frac{1}{100} \sum_{i=1}^{100} \left| \frac{q_{90\%}^{(i),\text{LSMC/NN}} - q_{90\%}^{\text{true}}}{q_{90\%}^{\text{true}}} \right| .$$

- The **true value** is derived from a nested Monte-Carlo simulation with $100\,000$ outer and $100\,000$ inner simulations.

Error measurement

- We determine the **90th percentile** $q_{90\%}^{\text{LSMC/NN}}$ of the annuity values for NN and LSMC.

- We compare the two procedures to the *true value* considering the **mean absolute percentage error** (MAPE):

$$\text{MAPE}^{\text{LSMC/NN}} = \frac{1}{100} \sum_{i=1}^{100} \left| \frac{q_{90\%}^{(i),\text{LSMC/NN}} - q_{90\%}^{\text{true}}}{q_{90\%}^{\text{true}}} \right| .$$

- The **true value** is derived from a nested Monte-Carlo simulation with $100\,000$ outer and $100\,000$ inner simulations.

Error measurement

- We determine the **90th percentile** $q_{90\%}^{\text{LSMC/NN}}$ of the annuity values for NN and LSMC.

- We compare the two procedures to the *true value* considering the **mean absolute percentage error** (MAPE):

$$\text{MAPE}^{\text{LSMC/NN}} = \frac{1}{100} \sum_{i=1}^{100} \left| \frac{q_{90\%}^{(i),\text{LSMC/NN}} - q_{90\%}^{\text{true}}}{q_{90\%}^{\text{true}}} \right| .$$

- The **true value** is derived from a nested Monte-Carlo simulation with $100\,000$ outer and $100\,000$ inner simulations.

Setup for LSMC and NN

- For the two regression techniques, we rely on a **small nested simulation** of 1000 outer and 10 inner simulations.
- We then do a **regression** on the $1000 \cdot 10 = 10\,000$ realisations.
- Then we **estimate** the quantile $q_{90\%}^{\mathrm{LSMC/NN}}$ on an **additional set** of $1\,000\,000$ outer scenarios.

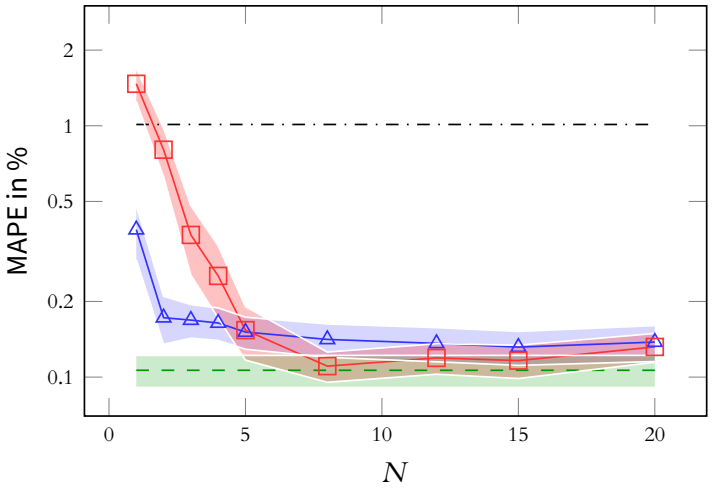
Setup for LSMC and NN

- For the two regression techniques, we rely on a **small nested simulation** of 1000 outer and 10 inner simulations.
- We then do a **regression** on the $1000 \cdot 10 = 10\,000$ realisations.
- Then we **estimate** the quantile $q_{90\%}^{\text{LSMC/NN}}$ on an **additional set** of $1\,000\,000$ outer scenarios.

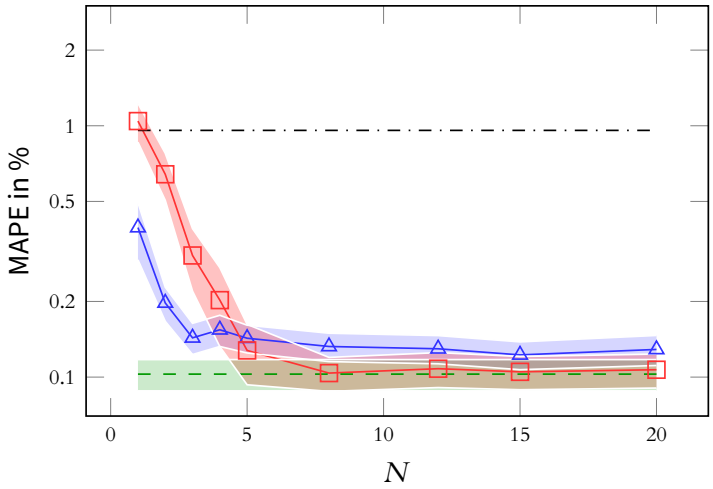
Setup for LSMC and NN

- For the two regression techniques, we rely on a **small nested simulation** of 1000 outer and 10 inner simulations.
- We then do a **regression** on the $1000 \cdot 10 = 10\,000$ realisations.
- Then we **estimate** the quantile $q_{90\%}^{\mathrm{LSMC/NN}}$ on an **additional set** of $1\,000\,000$ outer scenarios.

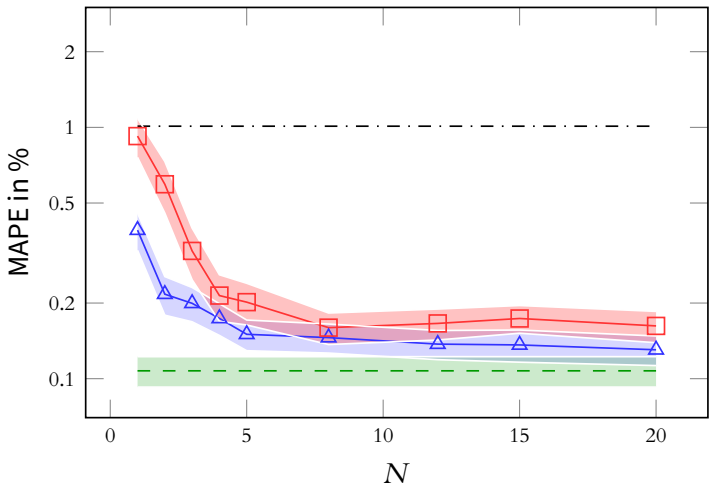
Numerical Results



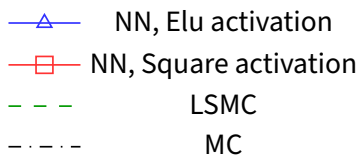
(a) loss function: Mean Squared Error



(b) loss function: Huber



(c) loss function: Mean Absolute Percentage Error



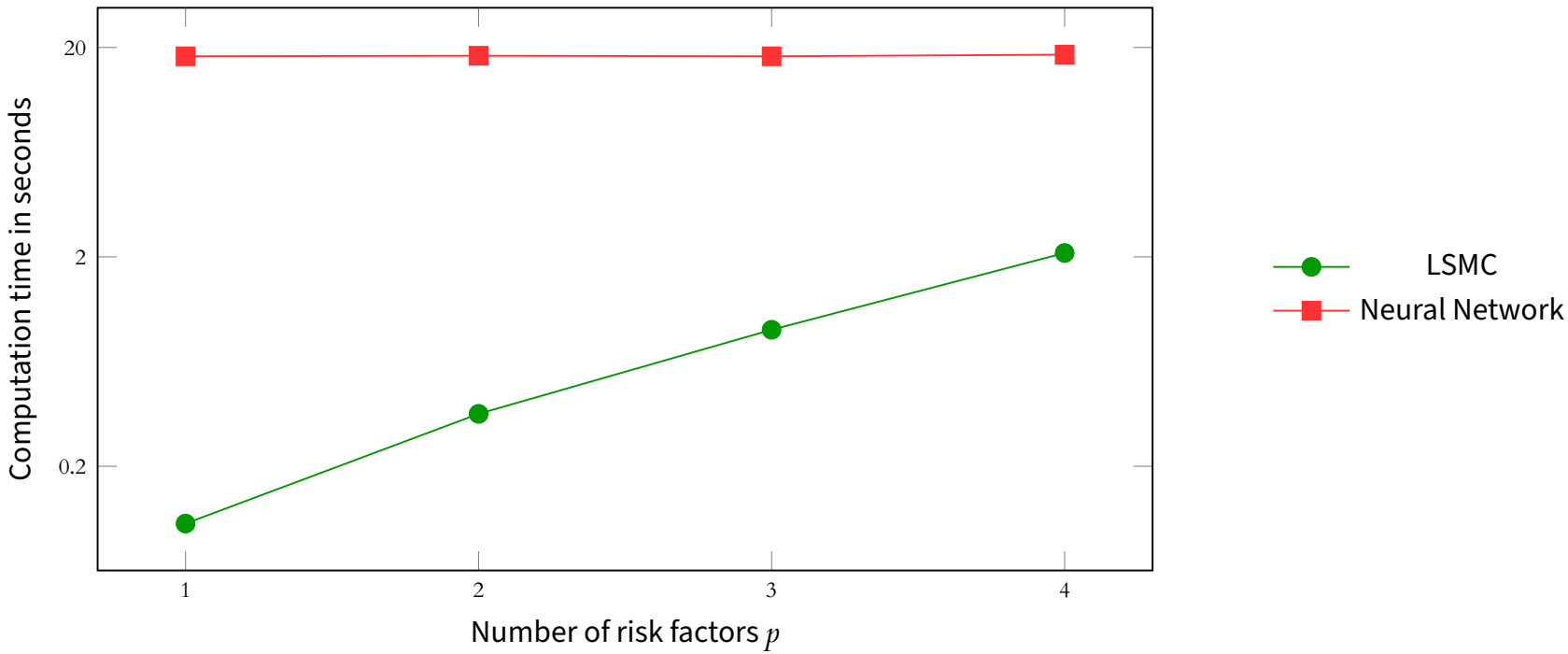
ETH zürich

Comparison NN and LSMC

e.g. Castellani et al. (2021): We need three-layer networks with many neurons to slightly beat LSMC. LSMC performs very well.

- We confirm this result.
- We provide small networks that lead to a similar performance as LSMC, but cannot beat it.
- The link between NN and LSMC may be a starting point to improve LSMC.

Curse of dimensionality



Summary

Takeaways

- A simple polynomial **LSMC is difficult to beat** by neural networks (in this application)
- We link LSMC and neural networks, allowing a slight extension of LSMC, still keeping some interpretability (small neural networks). We interpolate between LSMC and NN - **LSMC is a special case of NN**.
- Neural networks do not suffer from the **curse of dimensionality** in terms of the number of risk factors. LSMC requires a selection of relevant basis functions / interaction terms.
- Combination of **squared activation** function and **Huber loss** works best.

References i

Bacinello, A. R., Millossovich, P., and Viviano, F. (2021). An efficient Monte Carlo based approach for the simulation of future annuity values.

Bacinello, A. R., Millossovich, P., and Viviano, F. (2024). An iterative least-squares monte carlo approach for the simulation of cohort based biometric indices. *European Actuarial Journal*.

Barigou, K. and Delong, L. (2022). Pricing equity-linked life insurance contracts with multiple risk factors by neural networks. *Journal of Computational and Applied Mathematics*, 404:113922.

Bauer, D., Bergmann, D., and Reuss, A. (2010). Solvency II and nested simulations–a least-squares Monte Carlo approach.

Biffis, E. and Millossovich, P. (2006). The fair value of guaranteed annuity options. *Scandinavian Actuarial Journal*, 2006(1):23–41.

Cairns, A. J. (2011). Modelling and management of longevity risk: Approximations to survivor functions and dynamic hedging. *Insurance: Mathematics and Economics*, 49(3):438–453.

Castellani, G., Fiore, U., Marino, Z., Passalacqua, L., Perla, F., Scognamiglio, S., and Zanetti, P. (2021). Machine learning techniques in nested stochastic simulations for life insurance. *Applied Stochastic Models in Business and Industry*, 37(2):159–181.

Cheridito, P., Ery, J., and Wüthrich, M. V. (2020). Assessing asset-liability risk with Neural Networks. *Risks*, 8(1):16.

Dowd, K., Blake, D., and Cairns, A. J. G. (2011). A computationally efficient algorithm for estimating the distribution of future annuity values under interest-rate and longevity risks. *North American Actuarial Journal*, 15(2):237–247.

Fernandez-Arjona, L. (2021). A neural network model for solvency calculations in life insurance. *Annals of Actuarial Science*, 15(2):259–275.

References ii

Fernandez-Arjona, L. and Filipović, D. (2022). A machine learning approach to portfolio pricing and risk management for high-dimensional problems. *Mathematical Finance*, 32(4):982–1019.

Floryszczak, A., Le Courtois, O., and Majri, M. (2016). Inside the Solvency 2 black box: Net asset values and Solvency capital requirements with a least-squares Monte-Carlo approach. *Insurance: Mathematics and Economics*, 71:15–26.

Gan, G. (2013). Application of data clustering and machine learning in variable annuity valuation. *Insurance: Mathematics and Economics*, 53(3):795–801.

Graf, S. and Korn, R. (2020). A guide to Monte Carlo simulation concepts for assessment of risk-return profiles for regulatory purposes. *European Actuarial Journal*, 10(2):273–293.

Günther, S. and Hieber, P. (2024). Efficient simulation and valuation of equity-indexed annuities under a two-factor G2++ model. *European Actuarial Journal*, 14(3):905–928.

Ha, H. and Bauer, D. (2022). A least-squares Monte Carlo approach to the estimation of enterprise risk. *Finance and Stochastics*, 26(3):417–459.

Hejazi, S. A. and Jackson, K. R. (2017). Efficient valuation of SCR via a neural network approach. *Journal of Computational and Applied Mathematics*, 313:427–439.

Jonen, C., Meyhöfer, T., and Nikolić, Z. (2023). Neural networks meet least squares monte carlo at internal model data. *European Actuarial Journal*, 13(1):388–425.

Krah, A.-S., Nikolić, Z., and Korn, R. (2020). Least-squares monte carlo for proxy modeling in life insurance: neural networks. *Risks*, 8(4):116.

Li, N. and Lee, R. (2005). Coherent mortality forecasts for a group of populations: An extension of the Lee-Carter method. *Demography*, 42(3):575–594.

Longstaff, F. A. and Schwartz, E. S. (2001). Valuing American options by simulation: A simple least-squares approach. *Review of Financial Studies*, 14(1):113–147.