# A Multi-Agent Non-Stochastic Economic Simulator

Ulrich Wolffgang

University of Münster, Germany
ulrich.wolffgang@uni-muenster.de

**Abstract.** We introduce an agent-based computational economics simulator constituted by agent types *household*, *factory*, *trader*, *credit bank*, *central bank* and *state*. Contributions are support for arbitrary numbers of good types and currencies, as well as non-stochastic iterative algorithms for implementing agent decision making on pricing, utility and production optimization according to traditional marginalism. Additionally, best practices for development of agent-based economic simulators are presented.

**JEL Classification Codes:** C63, C68, D58

## 1 INTRODUCTION

Agent-based computational economics (ACE) promises microfoundation of macroeconomic models, with macroeconomic phenomena and aggregates emerging from micro agent behavior. In this paper we introduce an ACE simulator, which provides a Java modeling framework and implements an Arrow-Debreu-like [2] general equilibrium model according to neoclassical theory and traditional marginalism. Even though stochastic decision making is avoided on the microeconomic level and agent behaviors are minimal, complex oscillation patterns attracted to macroeconomic equilibria are induced by economic feedback cycles of the system.

In Section 2 related work in the ACE field is discussed. The rest of the paper is structured according to the Dahlem ABM documentation guidelines [18]. An overview of the scope, agent types and activities is given in Section 3. Design concepts of the economic model and the technical platform are presented in Section 4. Characteristic algorithms for pricing and maximization of utility and production output are specified in Section 5, providing implementations of iterative decision making under the paradigm of marginalism. Section 6 lists best practices gathered during the development process. In Section 7, we conclude and point out future work.

## 2 RELATED WORK

Recent implementations of agent-based economic models in the ACE field are:

The *Eurace Project*[1] is an agent-based macroeconomic simulation focussing on policy measures, which is implemented in C and XML utilizing the FLAME framework[2]. It consists of agents interacting on markets for labour, consumption goods and investment goods [4][5][6], an energy sector [13], a financial market, credit market and public sector including a government and central bank [3]. Agent behavior includes inventory-based production planning for a consumption good, pricing decisions determined by production costs and elasticity of demand, savings, and stochastic consumption planning of households [4].

[7] present an agent-based model constituted by workers i.e. households, consumption good firms, investment good firms, a commercial bank, a central bank and a government for taxation. Microeconomic decision making is partly stochastic and executed in a fixed sequential order. The model focusses on effects of Keynesian policy making on growth, income distribution and demand.

[11][12] specifies an agent-based simulation with focus on a decentralized and inventory-based pricing strategy. Homogenous agents produce, exchange, consume and price goods in a fixed sequential order of phases. Prices are modeled as private information of homogenous agents and aggregated to a public equilibrium price in a stochastic process. A considerable extension is the *Lagom* model [16]. Enhancements are the more specific agent types household and firm, capital accumulation, Schumpeterian growth, taxation, three economic sectors, a financial system providing a key interest rate according to the Taylor rule, and backtesting against statistical data of the German economy. Further extensions generalize the spatial aspect to multiple economic sectors [24].

*Jamel*[3] is an agent-based computational macroeconomic model implemented in Java, which is constituted by households for consumption, firms for production and one bank providing a currency in form of credit money [20][19]. Each time period is structured by a fixed sequential order of interactions such as wage and dividend payments, production of a single good, consumption and market interactions. Market order selection is stochastic as well as inventory-based production and pricing decision making [20].

# 3 OVERVIEW

## 3.1 RATIONALE

In the following we introduce an ACE simulator implemented in Java under an open source license[4]. Currently, a model akin to the Arrow-Debreu model [2] is implemented, which adheres to neoclassical microeconomic theory, including polypoly markets perpetuated by agent market participants.

Contributions of this paper are:

[1] `http://www.eurace.org/`
[2] `http://www.flame.ac.uk/`
[3] `http://p.seppecher.free.fr/jamel/`
[4] Project repository: `http://github.com/uwol/ComputationalEconomy`

- The simulator supports an arbitrary number of good types and currencies, enabling trading for arbitrage.
- Algorithms are provided, which specify microeconomic agent decision making on pricing, utility optimization and production output maximization according to traditional marginalism. Marginal calculus is supported both for fixed price functions as well as step price functions and applied iteratively and non-stochastically.
- Best practices are listed for the development of agent-based economics simulators.

### 3.2 AGENTS

The economic model of the simulator is constituted by the homogenous agent types *household*, *factory*, *trader*, *credit bank*, *central bank* and *state*. At simulation startup an exogenously defined number of agents is instantiated. Technically, all agent types are implemented as lifecycle entities, enabling dynamic instantiation and deconstruction of agents at runtime. This includes a controlled cancellation of business relations such as revocation of issued bonds and closing of bank accounts.

Agents are structured in an object-oriented type hierarchy. The root node of the taxonomy is defined by the abstract Java class *Agent*, which is specialized by classes for previously mentioned agent types. As factories, traders and banks are joint stock companies, they specialize the *Agent* class via an intermediate class *JointStockCompany*, which specifies behavior for dividend calculation and transfer.

### 3.3 Other entities

Non-agent entities of the microeconomic model are *goods*, *bank accounts*, *balance sheets*, *market orders*, *shares* and *bonds*. The latter two are generalized to the general concept of *property* and as such part of a property rights system, thus owned by agents and transferable between those. Comparably to agents, *properties* are designed as lifecycle entities.

Macroeconomic entities are *currencies*, *markets*, *national economies* and *sectors* within those. Each agent is associated to one of multiple national economies. A national economy is subdivided into economic sectors, one for each modeled good type (e. g. *coal*, *iron*, *energy* and *wheat*). The set of good types can be extended to arbitrary extent by the modeler, but is static at runtime. Each factory is specialized on one good type and thus associated to one sector. In these national economies microeconomic decisions of agents constitute macroeconomic aggregates such as *saving*, *consumption*, *production*, *utility*, *credit utilization* and *money supply*.

### 3.4 Boundaries

The simulation runs autonomously without inputs at runtime. However exogenous shocks such as sudden growth of productivity or deficit spending can be triggered programmatically.

### 3.5 Relations

Interaction between agents is intermediated by markets primarily, which are implemented as settlement markets and thus automatically enforce the delivery of goods and securities for payments. In addition, bilateral interactions between agents are implemented for decentralized activities such as dividend payments, bank account opening and emission of bonds by credit banks and states.

Agent interactions are either national or international, leading to flow of capital and goods between national economies. As each national economy has its own currency, currency markets exist, which based on supply and demand represent the productivity of national economies.

### 3.6 Activities

Mentioned agent types are characterized by following activities:

**Households** offer labour hours and receive utility from goods consumption based on CES or Cobb-Douglas utility functions. Utility maximization includes allocation of a daily time budget to labour and leisure time according to microeconomic theory. Households are price setters for the production factor *labour hour*.

$$y_{h,t} = y_{h,t}^{wage} + y_{h,t}^{div} + y_{h,t}^{int} + y_{h,t}^{trans} \tag{1}$$

Household income is composed of wage, dividends, interest and state transfers. Depending on the exogenous configuration, they make retirement savings from their income based on intertemporal consumption and retirement saving preferences, which are modeled by Irving-Fisher [8] and Modigliani [17] intertemporal choice models.

**Factories** produce goods as single-product companies according to an input-output-model, which is based on Cobb-Douglas, CES and root production functions. Inputs and outputs of this model are aforementioned good types, which hence serve either as consumption goods for households or as input factors for further processing. Factories are price setters for their produced good.

$$y_{f,t} = y_{f,t}^{rev} + y_{f,t}^{int} \tag{2}$$

Factory revenues are composed of sales revenues and interest. Depending on the exogenous configuration, capital is generated according to the *Solow-Swan* model [22] by buying investment goods of endogenously produced good

type *machine*. As goods are sold to households and households earn wages and dividends from factories, an economic cycle is established. The required liquidity for monetary transactions originates from credit money provided by credit banks.

**Credit banks** manage bank accounts of their customers and provide endogenous money in form of credit. Credit money is backed by central bank money according to predefined minimum reserve requirements. Depending on the exogenous configuration households make retirement savings in form of deposits on passive bank accounts. According to fractional reserve banking, credit banks invest those assets into bonds issued by state agents. Additionally, in case of multiple national economies credit banks trade currencies on foreign exchange markets, thus providing liquidity to trader agents.

$$0 = \sum_{a=1}^{N^a} m_{a,t}^{trans} + \sum_{a=1}^{N^a} m_{a,t}^{sav} \tag{3}$$

Bank account balances sum up to zero, typically with firms and states being debtors and households being creditors.

**Traders** import goods as price takers from foreign markets into their national economy for arbitrage. Goods are bought from foreign factories for foreign currency, which is exchanged with credit banks on currency markets. The resulting foreign exchange rates represent the productivity of national economies.

**Central banks** manage bank accounts of credit banks in their national economy. Aforementioned minimum reserve requirements of credit banks are fulfilled by lending central bank money against bonds issued by credit banks as security. Additionally, central banks monitor prices on good markets for calculating a price index. Deviations from an inflation-adjusted price index lead to adjustments of the central bank's key interest rate. Key interest rates have a reverse feedback on good market prices via a monetary transmission mechanism, as credit takers adjust the credit volume and thereby budget according to the key interest rate. Thus expansionary (contractionary) monetary policy induces growing (shrinking) budgets and thereby rising (falling) prices on markets.

**States** issue bonds, which are bought by credit banks for fractional reserve banking. Thereby national debt represents retirement savings of households. States receive seigniorage from their central bank, which is transferred to households, preventing hoarding of money.

Each period agents publish balance sheets, which are aggregated to sector and subsequently national balance sheets. Additionally, financial transactions are aggregated into a periodic monetary transactions adjacency matrix.

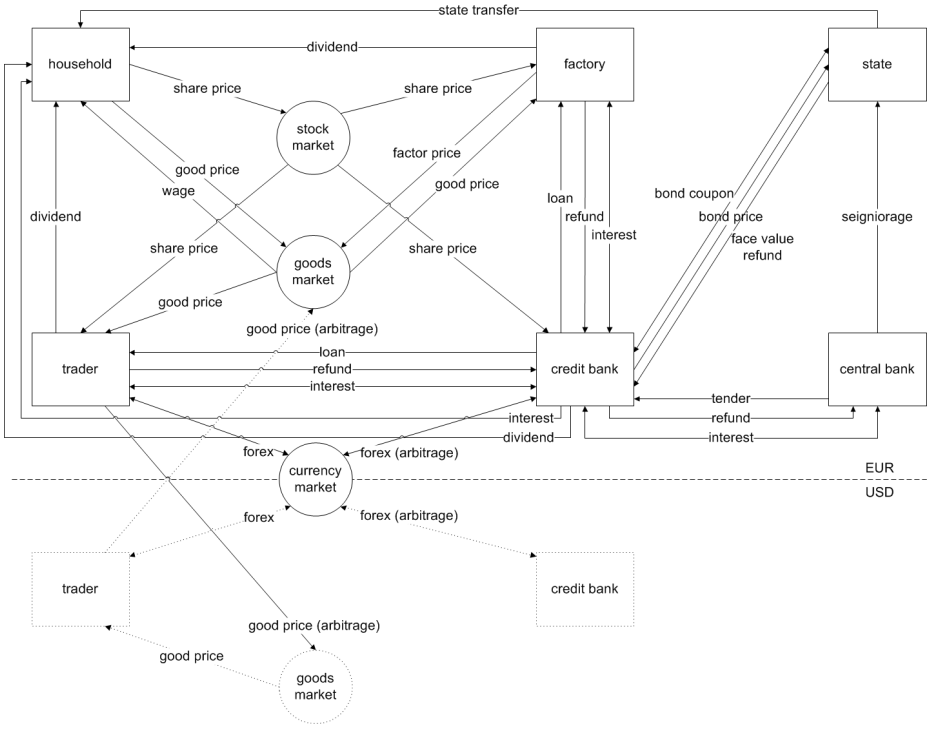


**Fig. 1.** financial flows

## 4 Design concepts

### 4.1 Time, activity patterns and activation schemes

The simulator is implemented as a Java application, following the object-oriented programming paradigm similarly to most agent-based modeling systems [1]. It is structured as a multi-tier architecture [9], consisting of a persistence tier implementing the microeconomic model in form of entities, a service tier implementing the simulator engine and socio-economic framework including markets and a property register, and a dashboard and API for user and software interaction. The engine can be run standalone without graphical output and is configured by configuration files, which bundle the exogenous parameters of the economic system. The preferred execution mode is in-memory for performance reasons, for completeness however SQL databases are supported, too.

Agents and other entities are implemented in the persistence tier as Hibernate entities. However in contrast to conventional entity beans, they not only represent state, but contain additional inner Java classes, which implement events. Agent behavior is implemented in those events such as *DailyLifeEvent*, *ProductionEvent* and *PayDividendEvent*. Time events are registered at the time system according to an observer pattern [10]. The time system proceeds hour-wise in

time and triggers all events registered for that point in time in random order, so that concurrent events are treated equally over time.

Events can be registered and removed from the time system at runtime, so that new time system observers such as agents can be instantiated and deconstructed endogenously at runtime by the simulation. Additionally, events can be associated to arbitrary points in time, avoiding an artificial order of economic execution phases for e. g. production or consumption.

### 4.2 Interaction protocols and information flows

Interaction protocols are specified in form of Java interfaces, which decouple implementations of markets, banks etc. from clients such as market participants or bank customers. Similarly to the Spring framework a central application context is provided, which binds Java classes to those Java interfaces, thereby realizing the principle of dependency injection [15]. E. g. the application context contains a reference to a specific market implementation, which is offered for interaction with agents under the general market interaction protocol i. e. market Java interface.

In each national economy a commodity polypoly market exists for every good type. Additionally, financial markets enable trading of currencies and properties, i. e. bonds and shares. Markets are implemented as perfect with free competition, no frictional costs or restrictions, homogeneous products, perfect information, perfect factor mobility and no costs of market entry [23]. Each market is structured as a collection of ask market orders, which are entered by suppliers and ordered by their price per unit, starting with the lowest price. Additionally, markets are specialized as settlement markets, which automatically transfer ownership of goods and money.

The market interaction protocol for buyers allows either execution of buy orders, optionally constrained by amount and price, or price requests for marginal prices, average prices and price functions. Both types of requests are responded by markets by calculating an optimal fulfillment set, which represents the optimal solution at the lowest total price. Price functions serve as a solution for the problem, that on the one hand markets manage discrete market orders with discrete prices, and on the other hand continuously differentiable price functions are needed due to the neoclassical nature of the simulation. Primarily this is the case for formulating Lagrange functions, so that utility or production maximization problems can be calculated under budget restrictions according to marginal calculus.

As each market is a nexus for the economic sector in its national economy, a high-performance and memory-efficient implementation is crucial for overall simulator performance. An important implementation aspect is, that agents can iterate in-memory over market orders into the depth of the market, beginning with the lowest price per unit, reducing performance impacts from aforementioned iterative analytical optimization calculations.

### 4.3 Forecasting

Decision making is parametrized with data from past periods, the agent's current financial status and the market situation. E. g. production planning of factories is based on current marginal costs and marginal revenues of goods, including a safety margin for assuring profitability. Likewise, pricing decisions are derived from sold and offered amounts in the last periods.

### 4.4 Behavioral Assumptions and Decision Making

Decision making of agents is bounded rational [21] and in contrast to related work neither inventory-based nor stochastic.

Factories maximize production according to traditional marginalism. They are bounded rational, as they do not take foreign goods markets into consideration. The rationale for this behavior is an efficient-market hypothesis, as traders import and export goods for arbitrage, providing convergence between national productivity and exchange rates.

Households optimize their utility by allocating their daily time between leisure and work and allocating income on goods for utility maximization. However they act bounded rational, as they first of all pay regard only to marginal prices without consideration of market depth and second do not observe foreign good and labour markets.

### 4.5 Learning

Decision making processes of agents are hard-coded in Java and therefore static. Learning in form of productivity growth is expressed by conventional parametrization of production function coefficients.

### 4.6 Population Demography

As all agents are lifecycle entities, a dynamic population size at runtime can be modeled. The economic model is designed with a varying number of households, which create further households in case of utility oversupply, and contrarily are deconstructed in case of ongoing undersupply or reaching a predefined age.

### 4.7 Levels of Randomness

The microeconomic behavior of agents at runtime is implemented non-stochastic. Apart from that, there are two sources of randomization.

First of all, the agent population can be initialized utilizing random number generators for parameters such as age of households and time system event registration. However this is optional and can be substituted by explicit configuration.

Secondly, implicit randomization occurs on a macroeconomic level in ambiguous situations. Such are markets with multiple market orders for the same good and price per unit, or undetermined execution orders for multiple events registered in the time system for the same moment in time.

### 4.8 Technical Architecture and Development

Microeconomical activities are logged into statistical data models, which enable introspection of individual agents as well as calculation of macroeconomic aggregates. Additionally, the mechanics of pivotal microeconomic decision making processes such as pricing behaviors are aggregated, giving insight into causes of e. g. market-wide price changes.

The front end is designed as a dashboard based on *JFreeChart*[5] and a separate API implemented according to the Java Management Extensions (JMX) standard. The latter enables network communication with the calculation engine and statistical data models, so that remote execution of the simulation is possible. As a special mode of server-side execution an optimization runner for ceteris paribus analysis is included.

The development of the simulator is conducted test-driven. JUnit[6] tests are defined for individual mathematical components such as utility functions as well as complex constellations such as market interactions. The laboratory-like tests are complemented by assertion statements that enforce a valid simulation state in form of constraints and proactively signal irregular states, avoiding consecutive faults.

In the development process Java has shown to be advantageous for debugging issues due to its virtual machine. Generally, it is advisable to choose a managed platform, which allows introspection into the system memory and execution paths via monitoring tools such as VisualVM.

## 5 Functional Specification

Following agent behaviors for pricing, utility maximization and production optimization are designed according to marginal calculus. They relate microeconomic calculation results to actions and induce reproducibly, that according to classical microeconomic theory a macroeconomic equilibrium emerges. The equilibrium manifests in steady prices, production plans etc. across markets and national economies. In case of exogenous stimulus the system adapts to a new equilibrium automatically. The rate of adaption is determined by microeconomic adaption rates in form of price change increments and credit expansion rates.

### 5.1 Pricing Behavior

The pricing behavior implements decision making on the optimal agent-individual price for a good type in a period.

Related work focusses on inventory-based pricing behaviors [12][20][4] for agent-individual prices, which are aggregated to a macroeconomic equilibrium price, partly orchestrated in a stochastic process. In contrast, the following pricing behavior is parameterized with offered and sold amounts of the last and penultimate periods.

[5] `http://www.jfree.org`
[6] `http://junit.org`

In detail the decision making process is specified in pseudo code in listing 1. Price changes depend on, whether nothing (everything) has been sold in the last period, or less (more) has been sold in the last period in comparison to the penultimate period. A general design principle is to assure structural symmetry between constellations leading to higher prices and such leading to lower prices. This principle is complemented by a second design principle of conducting price changes only, when clear decision indicators are present, or vice versa to keep the price unchanged, when no such significant indicators are perceivable by the agent.

Exceptions from these principles are:

**Situations of limited market clearing** for the priced good type necessitate a deviation from the symmetry design principle. Such situations are expressed by stocks of that good type at inventories of agents. However, inventories by definition have to be non-negative, inducing a bias towards oversupply and market depth and against undersupply. For compensation, the pricing behavior has to react asymmetrically on such constellations and reduce limited market clearing, until inventories are cleared. This is achieved by lowering prices under situations of limited market clearing. It has shown, that a sufficient reaction pattern for such situations is to decrease the price when nothing has been sold in the last period. Under a situation of limited market clearing the constellation of selling nothing for multiple periods is significantly more likely than the constellation of selling everything.

**Slight implicit upward price pressure** is reasonable in cases, when no explicit price change decision is made. This is according to the natural motivation of sellers to raise prices for own benefit. Additionally, from a systematic perspective, a slight implicit upward price pressure is helpful due to the economical asymmetry, that typical key interest rates are non-negative. Therefore monetary policy is limited in cases of deflation corresponding to real-world scenarios, but unlimited in cases on inflation, making rising prices i. e. inflation more manageable from the view of the central bank and a self-adjusting simulation.

The decision for the price change direction is complemented by a decision on the magnitude of the price movement (listing 2).

In case of a falling price a differentiation is made, whether the last decision resulted in a falling price, too (listing 3). In that case, the downward price movement is accelerated. Contrarily, a rising price as the outcome of the last decision signals an oscillating price, resulting in an slowdown of price movement. In case of an oscillation around a price equilibrium this induces a gradually decreasing amplitude of oscillation.

**Listing 1.** pricing behavior - direction

```
calculateNewPrice(offeredAmountInLastPeriod, offeredAmountInPenultimatePeriod
    , soldAmountInLastPeriod, soldAmountInPenultimatePeriod,
    priceInLastPeriod, priceInPenultimatePeriod){
  // nothing sold?
  if offeredAmountInLastPeriod > 0 & soldAmountInLastPeriod <= 0
    return calculateDecreasedPriceExplicit(priceInLastPeriod,
        priceInPenultimatePeriod);

  // everything sold?
  if offeredAmountInLastPeriod > 0 & soldAmountInLastPeriod =
      offeredAmountInLastPeriod
    return calculateRaisedPriceExplicit(priceInLastPeriod,
        priceInPenultimatePeriod);

  // sold less?
  // something was offered last period and something was sold in the
      penultimate period
  // and there was sold less in last period than in the penultimate period
  // and there was offered more in last period than sold in penultimate
      period
  // -> there was a chance in the last period to outperform the amount sold
      in the penultimate period
  if offeredAmountInLastPeriod > 0 & soldAmountInPenultimatePeriod > 0 &
      soldAmountInLastPeriod < soldAmountInPenultimatePeriod &
      offeredAmountInLastPeriod >= soldAmountInPenultimatePeriod
    return calculateDecreasedPriceExplicit(priceInLastPeriod,
        priceInPenultimatePeriod);

  // sold more?
  // something was offered last period and something was sold in the
      penultimate period
  // and there was sold more in last period than in the penultimate period
  // and there was offered more in the penultimate period than sold in the
      last period
  // -> there was a chance in the penultimate period to outperform the amount
       sold in the last period
  if offeredAmountInLastPeriod > 0 & soldAmountInPenultimatePeriod > 0 &
      soldAmountInLastPeriod > soldAmountInPenultimatePeriod &
      offeredAmountInPenultimatePeriod >= soldAmountInLastPeriod)
    return calculateRaisedPriceExplicit(priceInLastPeriod,
        priceInPenultimatePeriod);

  // implicit pricing pressure
  return calculateRaisedPriceImplicit(priceInLastPeriod);
}
```

**Listing 2.** pricing behavior - magnitude

```
var priceFactor

calculateDecreasedPriceExplicit(priceInLastPeriod, priceInPenultimatePeriod){
  priceFactor = calculatePriceDecreasingFactor(priceInLastPeriod,
      priceInPenultimatePeriod);
  return priceInLastPeriod / (1 + priceFactor);
}

calculateRaisedPriceExplicit(priceInLastPeriod, priceInPenultimatePeriod){
  priceFactor = calculatePriceRaisingFactor(priceInLastPeriod,
      priceInPenultimatePeriod);
  return priceInLastPeriod * (1 + priceFactor);
}
```

**Listing 3.** pricing behavior - factor

```
const maxPriceFactor
// priceFactorChange is global constant > 1
const priceFactorChange

calculatePriceDecreasingFactor(priceInLastPeriod, priceInPenultimatePeriod){
  // price falling since two periods
  if priceInLastPeriod < priceInPenultimatePeriod
    return min(maxPriceFactor, priceFactor * priceFactorChange);

  // price oscillating
  if priceInLastPeriod > priceInPenultimatePeriod
    return priceFactor / priceFactorChange;
}

calculatePriceRaisingFactor(priceInLastPeriod, priceInPenultimatePeriod){
  // price rising since two periods
  if priceInLastPeriod > priceInPenultimatePeriod
    return min(maxPriceFactor, priceFactor * priceFactorChange);

  // price oscillating
  if priceInLastPeriod < priceInPenultimatePeriod
    return priceFactor / priceFactorChange;
}
```

### 5.2 Utility Optimization Behavior

In the economic model of the simulation social welfare is expressed by cardinal utility of households. Each period households maximize their period utility under a given budget. Arbitrary utility functions can be chosen at simulation initialization, as long as they implement a Java interface with basic operations for calculating a function value and the partial derivate for a set of inputs with respect to a specific input.

Currently, three complementary approaches for solving an utility maximization problem under a given budget and market price function are implemented:

**Analytical** An analytical solution is calculated in cases, when the utility function provides an operation, which specifies the optimal selection of inputs under given budget and prices. Such an operation has to be derived ex ante by solving the corresponding conventional Lagrange optimization problem and is specific to the utility function, hence hard-coded. Advantages are fast execution and a perceptible conformance to neoclassical theory. However, problems arise in context of aforementioned markets with distinct market orders. Markets with discrete market orders induce non-continuous marginal price step functions due to the fact, that each market order is associated to a constant price per unit. Correspondingly, the price function has discontinuities, limiting applicability of analytical solving.

**Iterative** The appropriate parametrization of aforementioned Cobb-Douglas and CES functions ensures the existence of convex optimization problems. This enables the application of gradient ascent as a simple numeric heuristic due to the fact, that every maximum has to be global. In contrast to the instant analytical calculation, gradient ascent is implemented as an iterative

strategy by extending the bundle of inputs step by step until the budget restriction is violated (listing 4). Discontinuities of the market price function can be neglected, as differential calculus is avoided.

**Brute-force** Independently of convexity concerns of the optimization problem brute-force solving varies the function's inputs systematically, searching for input constellations that effect a global maximum. Due to exponential complexity with regard to the number of inputs, a scan of the complete domain is feasible only under a tight budget, a small number of inputs and a sufficiently coarse increment for varying inputs.

When market depth is limited or marginal market prices vary with demand, the analytical approach is infeasible. However in context of JUnit tests it serves as a benchmark for artificial scenarios of price functions with constant prices and unlimited market depth, as in those cases all three approaches should lead to identical results. In contrast, JUnit tests for more complex market situations are conducted by comparing outcomes of iterative and brute-force approaches. In both scenarios calculated optima are verified by the general criterion, that for each input the marginal output has to be identical, i. e. no mutual substitution of inputs offers a higher outcome.

The iterative implementation for traditional marginalism has to consider several aspects beyond classical theory (listing 4). Such are (1) situations of cleared input markets, although the function under optimization requires all inputs to be non-zero (e. g. Cobb-Douglas), (2) situations of limited market depth inducing substitution effects and (3) a heuristic for determining the optimal increment. Choosing a valid increment is mandatory due to the risk of abruptly stepping out of the solution space in an iteration, leading to termination of the algorithm. This especially is a problem in the initialization phase of the algorithm, when incrementing single inputs by absolute values results in significant changes relative to the other inputs.

### 5.3 Production Optimization Behavior

Comparably to utility functions, production functions are maximized under budget restrictions by solving convex optimization problems.

However, additional constraints have to be checked, namely the optional restriction of maximum output (listing 5) and the obligatory restriction of marginal profitability (listing 6). The latter according to marginalism demands that marginal revenue has to cover marginal costs when deciding for production of an additional unit. Depending on the configuration of the simulation marginal costs rise in consequence of diminishing returns and due to rising marginal prices. In contrast, marginal production declines in conformance to exogenously defined production functions.

The algorithm for solving convex production optimization problems is identical to the algorithm for utility maximization, however extended by evaluations for mentioned additional restrictions at the noted extension point in listing 4. In contrast to related work [20] the agent's inventory size of the produced good type

is not taken into account directly, but rather by scarceness on markets expressed in market prices.

**Listing 4.** iterative output maximization

```
calculateOutputMaximizingInputsIterative(inputTypes,
    priceFunctionsOfInputTypes, budget, numberOfIterations,
    initializationValue, pricesAreNaN,
    needsAllInputFactorsNonZeroForPartialDerivate){

  // (1) if some goods / prices are not available (NaN), but are required
  // -> return input bundle with zero inputs
  if pricesAreNaN & needsAllInputFactorsNonZeroForPartialDerivate
    return bundleOfZeroInputs

  // budget check
  if budget <= 0
    return bundleOfZeroInputs

  for (inputType : inputTypes)
    bundleOfInputs.put(inputType, initializationValue)

  budgetSpent = 0
  budgetPerIteration = budget / numberOfIterations

  while (true){
    // in case of overspending the budget
    if budgetSpent + budgetPerIteration > budget
      // terminate the while loop
      break

    optimalInputType = findHighestPartialDerivatePerPrice(bundleOfInputs,
        priceFunctionsOfInputTypes)

    // (2) if no optimal input type could be found, markets are sold out
    if optimalInputType = null
      // terminate the while loop
      break
    else{
      priceFunctionOfOptimalInputType = priceFunctionsOfInputTypes.get(
          optimalInputType)
      oldAmountOfOptimalInputType = bundleOfInputs.get(optimalInputType)
      marginalPriceOfOptimalInputType = priceFunctionOfOptimalInputType.
          getMarginalPrice(oldAmountOfOptimalInputType)

      // (3) additional amounts have to grow slowly to stay in solution space
      additionalAmountOfInputType = min(
        budgetPerIteration / marginalPriceOfOptimalInputType,
        max(oldAmountOfOptimalInputType, initializationValue)
      )

      bundleOfInputs.get(optimalInputType) += additionalAmountOfInputType

      // extension point for production restrictions (listings 5 & 6)

      budgetSpent += marginalPriceOfOptimalInputType *
          additionalAmountOfInputType
    }
  }

  // reset initialization values
  for (inputType : getInputTypes())
    bundleOfInputs.get(inputType) -= initializationValue

  return bundleOfInputs
}
```

**Listing 5.** maximal output restriction

```
newOutput = calculateOutput(bundleOfInputs)

if newOutput > maxOutput
  // revert amount of optimal good type
  bundleOfInputs.get(optimalInputType) = oldAmountOfOptimalInputType
  // terminate while loop
  break
```

**Listing 6.** marginal profit restriction

```
marginalOutputOfOptimalInputType = calculateMarginalOutput(bundleOfInputs,
    optimalInputType)
marginalPriceOfOptimalInputTypePerOutput = marginalPriceOfOptimalInputType /
    marginalOutputOfOptimalInputType

if estimatedMarginalRevenueOfGoodType <
    marginalPriceOfOptimalInputTypePerOutput
  // revert amount of optimal good type
  bundleOfInputs.get(optimalInputType) = oldAmountOfOptimalInputType
  // terminate while loop
  break
```

### 5.4 Evaluation

The presented algorithms are evaluated using an economic reference model, which is constituted as a single national economy with following agents:

- 1000 households with utility function

$$u = x_w^{0.4} * x_c^{0.4} * x_l^{0.2} \tag{4}$$

- Four factories in the *coal* sector with production function

$$x_c = 5 * x_l^{0.5} \tag{5}$$

- Four factories in the *wheat* sector with production function

$$x_w = 5 * x_l^{0.5} \tag{6}$$

- Five credit banks providing liquidity in form of credit money. Each factory is given at most 10.000 monetary units.
- One central bank controlling the price level via the key interest rate and monetary transmission mechanism, optionally limiting credit supply.

The simulation is executed twice, each time spanning three simulated years. In the first run, exogenous variables are not changed, resulting in an equilibrium that is perpetuated autonomously (solid lines in figures 2, 3, 4, 5).

In contrast, in the second run the productivity of the *wheat* sector is lowered exogenously during the second year by dividing the initial productivity factor 5.0 monthly by 1.05 (dotted line). Accordingly, *wheat* output is contracted due to the change in productivity, leading to rising *wheat* prices.

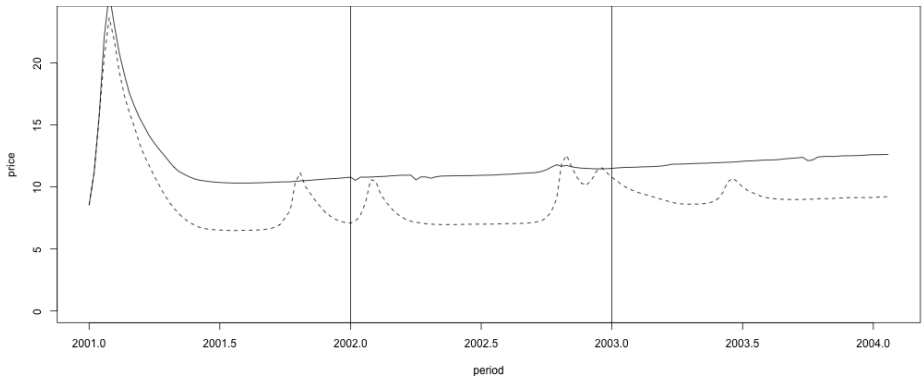


**Fig. 2.** avg. weekly *coal* price

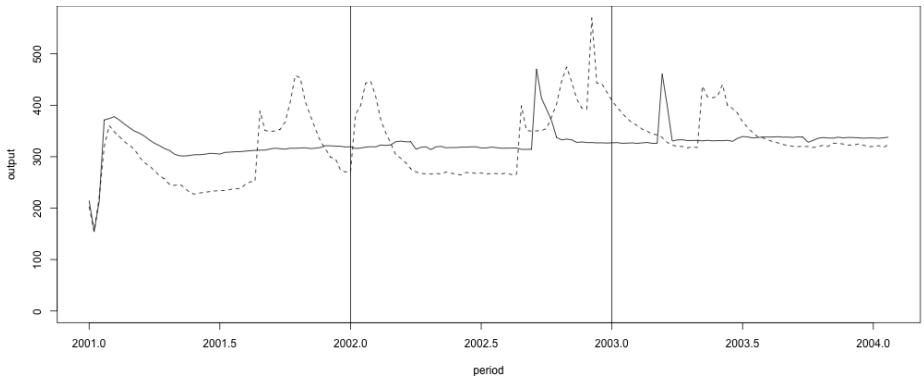


**Fig. 3.** avg. weekly *coal* output

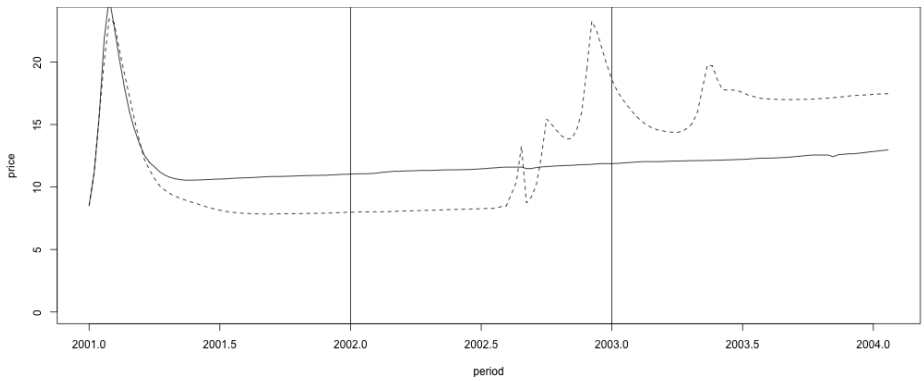


**Fig. 4.** avg. weekly *wheat* price

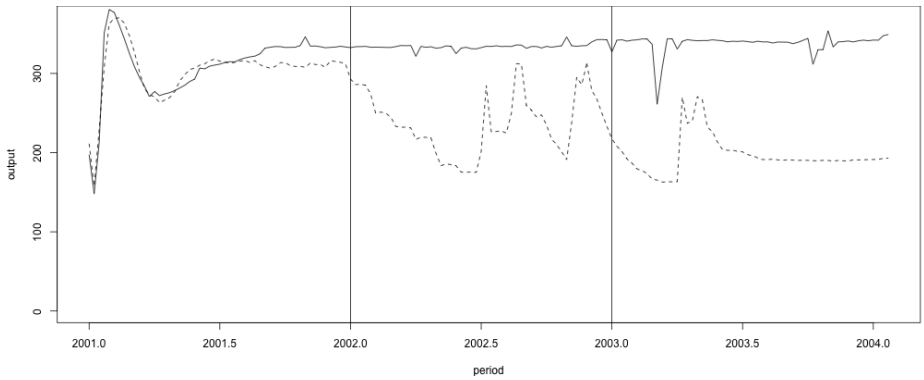


**Fig. 5.** avg. weekly *wheat* output

## 6 BEST PRACTICES

During the development process following best practices emerged:

**Assertions** Assertions specify valid system states in form of predicates. They check preconditions and postconditions of model state at runtime, and as such complement test-driven software development. In Java evaluation of assert statements is optional and depends on the VM configuration. Thereby it is possible to switch between either validation or performance oriented execution modes. The simulation engine makes extensive use of assertions, e. g. for ensuring compliance of calculations with budget constraints and correct execution of money transfers.

**Assurances** In contrast to assertions that lead to program termination when violated, assurances remediate invalid system states. In the simulation engine they are implemented as preconditions and ensured prior to most events. E. g. the *ProductionEvent* of factories assures the existence of a transaction bank account required for buying production factors. Advantages are that (1) the execution context i. e. required system state for events is specified explicitly, (2) agents react at runtime on irregular situations such as closed bank accounts and (3) system state is ensured as late as possible.

**Visualization of decision making mechanics** Most phenomena of the economic system such as rising prices are multicausal. Hence for analysis individual agent decisions have to be made comprehensible. While introspecting individual behaviors and decision making processes of agents can assist in such situations, it has shown that phenomena originating from interdependent stimuli and feedback cycles stay unclear. A solution is to aggregate and visualize mechanics of individual behavioral decision making processes to the modeler. E. g. execution path coverage of the pricing behavior is aggregated each period by summing up the strength of individual price movement. Finally, those aggregations are supplemented by the average decision and vi-

sualized in time series diagrams, giving an overview of pricing pressure on markets.

**Delegates** Most entities such as agents, bonds or bank accounts have a life cycle. This leads to memory management complexity, e. g. when references from agents to a deconstructed bank account have to be cleared. Thus agents are decoupled from such entities via delegates. This enables lazy evaluation e. g. of dividend bank accounts and decouples entities, making the simulation robust against extraordinary events such as insolvencies of banks.

**Arithmetic precision** Real numbers as the common number system of economic models are represented as floating point numbers in software systems, mostly according to the IEEE Standard for Floating-Point Arithmetic (IEEE 754) [14]. However, due to limited computing resources real numbers can only be approximated, necessitating conversion rules that are part of IEEE 754 and applied implicitly. When designing long-running closed-world simulations with a high number of calculations, *round to nearest* should be chosen in contrast to directed roundings, avoiding implicit shrinking or expansion of theoretically constant aggregates such as money supply. Additionally, valid arithmetic operations have to be chosen for the intended result number type. E. g. when dividing integers in Java, in advance a cast of at least one operand to data type *double* has to be made for ensuring an arithmetic operation with double precision. Otherwise implicit loss of precision takes place, leading to biased mathematical results and erratic systemic behavior.

## 7 CONCLUSION

We have introduced an ACE simulator implemented in Java. The current economic model is constituted by the agent types *household*, *factory*, *trader*, *credit bank*, *central bank* and *state*, as well as an arbitrary number of good types and currencies. The presented algorithms specify iterative and non-stochastic agent decision making on pricing, utility and production output maximization according to traditional marginalism. On the macroeconomic level reproducible equilibria and oscillating system behavior emerge, enabling ceteris paribus analysis. Additionally, we have presented several best practices for development of ACE simulators. The source code is accessible at the code repository under an open source license.

Future work includes efforts for (1) proper calibration of the economic model with given statistical data, (2) implementing the IS-LM model and (3) implementing the Taylor rule for central bank policy.

## References

1. Allan, R.: Survey of agent based modelling and simulation tools. Tech. rep. (Oct 7, 2010)
2. Arrow, K.J., Debreu, G.: Existence of an equilibrium for a competitive economy. Econometrica 22(3), 265–290 (Jul 1954)

3. Cincotti, S., Rabertoa, M., Teglio, A.: Credit money and macroeconomic instability in the agent-based model and simulator eurace. Economics: The Open-Access, Open-Assessment E-Journal 4(2010-26) (2012)
4. Dawid, H., Gemkow, S., Harting, P., Kabus, K., Wersching, K., Neugart, M.: Skills, innovation, and growth: An agent-based policy analysis. Journal of Economics and Statistics (2+3), 251–275 (Jun 2008)
5. Dawid, H., Gemkow, S., Harting, P., Neugart, M.: Spatial skill heterogeneity and growth: An agent-based policy analysis. Journal of Artificial Societies and Social Simulation 12(4) (2009)
6. Dawid, H., Neugart, M.: Agent-based models for economic policy design. Eastern Economic Journal 37, 44–50 (2010)
7. Dosi, G., Fagiolo, G., Napoletano, M., Roventini, A.: Income distribution, credit and fiscal policies in an agent-based keynesian model. Tech. rep., Laboratory of Economics and Management (LEM), Sant'Anna School of Advanced Studies, Pisa, Italy (Dec 23, 2012)
8. Fisher, I.: The Theory of Interest. Macmillan, New York (1930)
9. Fowler, M.: Patterns of Enterprise Application Architecture. Addison-Wesley, Amsterdam (Nov 15, 2002)
10. Gamma, E., Helm, R., Johnson, R., Vlissides, J.: Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, Amsterdam (Oct 31, 1994)
11. Gintis, H.: The emergence of a price system from decentralized bilateral exchange. The B.E. Journal of Theoretical Economics 6(1), 1–15 (2006)
12. Gintis, H.: The dynamics of generalized market exchange. Tech. rep. (Oct 13, 2010)
13. van der Hoog, S., Deissenberg, C.: Energy Shocks and Macroeconomic Stabilization Policies in an Agent-based Macro Model. Springer, Berlin (2011)
14. IEEE Task P754: Ieee standard for binary floating-point arithmetic. Tech. rep., New York (1985)
15. Johnson, R.: J2ee development frameworks. Computer 38(1), 107–110 (Jan 2005)
16. Mandel, A., Fürst, S., Lass, W., Meissner, F., Jaeger, C.: Lagom generic: an agent-based model of growing economies. Tech. rep. (Jan 2009)
17. Modigliani, F.: The life cycle hypothesis of savings, the demand for wealth and the supply of capital. Social Research 33, 160–217 (1966)
18. Sarah Wolf, Jean-Philippe Bouchaud, F.C.S.C.H.D.H.G.S.v.: Describing economic agent-based models - dahlem abm documentation guidelines (Aug 2012)
19. Seppecher, P.: Un modèle macroéconomique multi-agents avec monnaie endogène. Tech. rep., Groupement de Recherche en Economie Quantitative d'Aix-Marseille (Mar 2009)
20. Seppecher, P.: Jamel a java agent-based macroeconomic laboratory. Tech. rep., University of Nice Sophia-Antipolis (May 14, 2012)
21. Simon, H.A.: Simon aer 59. The American Economic Review 49(3), 253–283 (Jun 1959)
22. Solow, R.M.: A contribution to the theory of economic growth. Quarterly Journal of Economics 70(1), 65–94 (Feb 1956)
23. Stigler, G.J.: Perfect competition, historically contemplated. Journal of Political Economy 65(1), 1–17 (Feb 1957)
24. Wolf, S., Fürst, S., Mandel, A., Lass, W., Lincke, D., Meissner, F., Pablo-Marti, F., Jaeger, C.: Lagom regio - a multi-agent model of several economic regions. Environmental Modelling & Software 44, 25–43 (Jun 2013)