Deep Analysis · UID-Explore Engine

Stand 27.09.2025

Position im System

Die Engine ist der deterministische Rechenkern. Sie erhält vom **Director** einen konsolidierten Parametersatz und liefert als Ergebnis **Zeitreihen** der Modellkompartimente sowie Metadaten und einen Drift-Wert. Der Director veröffentlicht die Ergebnisse über den Bus an Chart, KPI und Vektorrad.

Schnittstellenpartner

- **Director (`uid.js`)** ruft `run({model, params, integrator})` auf.
- **Schema (`schema.js`)** liefert Katalog, Normalisierung und Kopplungen (z. B. $R_0 \leftrightarrow \beta \leftrightarrow \gamma$, $\gamma \leftrightarrow D$, $\sigma \leftrightarrow L$).
- **Event-Bus (`bus.js`)** transportiert `uid:e:model:update`, `uid:e:sim:data`, `uid:e:engine:status`, `uid:e:error`.

API der Engine

```
export function run({ model = 'SIR', params = {}, integrator = 'rk4' })
// → { series, meta, drift }
```

Inputs

- `model` Name des Modells in Großschreibung intern (`SIR`, `SEIR`, `SIRD`, `SIRV`; optional `SIS`).
- `params` konsolidierte Parameter inklusive `N`, `I0`, `T`, `dt`, `beta`, `gamma`, `sigma`, `mu`, `nu`, `measures` usw.
- `integrator` `euler`, `heun` oder `rk4`.

Outputs

- `series` Objekt mit Arrays gleicher Länge `steps+1`:
  - `t[k]` Zeit in Tagen
  - Je Kompartiment ein Array gemäß `meta.dims` (z. B. `S[k]`, `I[k]`, `R[k]`, …)
- `meta` `{ model, method, dims }`
- `drift` Betrag der Massenabweichung am Ende `|Σ(y_end) − N|`

Schutz und Limits

- `N ≥ 1`, `dt ≥ 1e-6`, `T ≥ dt`, `steps = floor(T/dt) ≥ 1`.

Interne Datenstruktur

- **Zustandsvektor** `y` in der Reihenfolge der Modell-`dims`.
- **Series** wird einmal angelegt: `{ t: new Array(steps+1), S: … }` und in der Schleife gefüllt.
- **Startwerte** aus `M.init({ ...params, N, I0 })`, negatives wird auf `0` gesetzt.

Integratoren

```
function stepEuler(f,p,y,h) { const k1=f(p,y); return
y.map((v,i)=>v+h*k1[i]); }
function stepHeun (f,p,y,h) { const k1=f(p,y); const
y2=y.map((v,i)=>v+h*k1[i]); const k2=f(p,y2); return
y.map((v,i)=>v+(h/2)*(k1[i]+k2[i])); }
function stepRK4 (f,p,y,h) { const k1=f(p,y); const
y2=y.map((v,i)=>v+(h/2)*k1[i]); const k2=f(p,y2); const
y3=y.map((v,i)=>v+(h/2)*k2[i]); const k3=f(p,y3); const
y4=y.map((v,i)=>v+h*k3[i]); const k4=f(p,y4); return
y.map((v,i)=>v+(h/6)*(k1[i]+2*k2[i]+2*k3[i]+k4[i])); }
```

`rk4` ist Standard. Der Integrator wird als Funktion `step(f, params, y, dt)` verwendet.

Zeitraster und Schleife

```
const steps = Math.max(1, Math.floor(T/dt));
series.t[0] = 0; series[dim][0] = y0;
for (let k=1; k<=steps; k++) {
  y = step(f, params, y, dt);
  // Non-Negativity
  for (let i=0;i<y.length;i++) y[i] = Math.max(0, y[i] || 0);
  // Sanfte Massenkorrektur
  const sum = y.reduce((a,b)=>a+b,0), drift = N - sum;
  if (Math.abs(drift) > Math.max(1e-9*N, 1e-9)) y[0] = Math.max(0,
y[0]+drift);
  // Ablage
  series.t[k] = k*dt; for (let i=0;i<M.dims.length;i++)
series[M.dims[i]][k] = y[i];
}
```

- **Non-Negativity** verhindert negative Bestände.
- **Sanfte Massenkorrektur** kompensiert numerische Drift auf `S` und hält die Summe nahe `N`.

Modelle und Gleichungen

Im Folgenden ist `β_eff = β · (1 − measures)` und `N` die fest vorgegebene Gesamtbevölkerung. Alle Raten sind pro Tag.

SIR

Zustand `y = [S, I, R]`, Start $S_0 = \max(0, N - I_0)$, I_0, $R_0=0$.

- $dS/dt = -\beta\_eff \cdot S \cdot I / N$
- $dI/dt = \beta\_eff \cdot S \cdot I / N - \gamma \cdot I$
- $dR/dt = \gamma \cdot I$

SEIR

Zustand `y = [S, E, I, R]`, zusätzliche Exponierte mit Rate σ.

- $dS/dt = -\beta\_eff \cdot S \cdot I / N$
- $dE/dt = \beta\_eff \cdot S \cdot I / N - \sigma \cdot E$
- $dI/dt = \sigma \cdot E - \gamma \cdot I$
- $dR/dt = \gamma \cdot I$

SIRD

Zustand `y = [S, I, R, D]`, krankheitsspezifische Mortalität μ.

- $dS/dt = -\beta\_eff \cdot S \cdot I / N$
- $dI/dt = \beta\_eff \cdot S \cdot I / N - \gamma \cdot I - \mu \cdot I$
- $dR/dt = \gamma \cdot I$
- $dD/dt = \mu \cdot I$

SIRV

Zustand `y = [S, I, R, V]`, Impfrate ν verschiebt `S → V`.

- $dS/dt = -\beta\_eff \cdot S \cdot I / N - \nu \cdot S$
- $dI/dt = \beta\_eff \cdot S \cdot I / N - \gamma \cdot I$
- $dR/dt = \gamma \cdot I$
- $dV/dt = \nu \cdot S$

SIS (optional)

Zustand `y = [S, I]`, Genesene werden wieder empfänglich.

- $dS/dt = -\beta\_eff \cdot S \cdot I / N + \gamma \cdot I$
- $dI/dt = \beta\_eff \cdot S \cdot I / N - \gamma \cdot I$

Kopplungen und abgeleitete Größen

Die Engine nutzt Parameter so, wie sie der Director liefert. Die Kopplungen passieren vorher über `schema.js`:

- **$R_0 \leftrightarrow \beta \leftrightarrow \gamma$** mit $\beta = R_0 \cdot \gamma$ und $\gamma = 1/D$.
- **$\sigma \leftrightarrow L$** mit $\sigma = 1/L$.
- **Measures** wirken auf die Ansteckungsrate via `β_eff`.

Start- und Randbedingungen

- Standardmäßig nur I_0 als expliziter Startwert, übrige Kompartimente starten bei 0 und $S_0 = N - I_0$.
- Alle Zustände werden bei jedem Schritt auf ≥ 0 geklemmt.
- Die Summe wird pro Schritt sanft auf N zurückgeführt.

Output-Gestalt

```
{
  series: {
    t: [0, dt, 2dt, …],
    S: [...], E: [...], I: [...], R: [...], D: [...], V: [...]
  },
  meta: { model: 'SIR'|'SEIR'|'SIRD'|'SIRV'|'SIS', method:
'euler'|'heun'|'rk4', dims: ['S','I','R', …] },
  drift: Number // |Σ(y_end) - N|
}
```

Konsumenten können sich an `meta.dims` orientieren und sind dadurch modellagnostisch.

Numerische Stabilität und Genauigkeit

- `rk4` ist für interaktive Anwendungen ein sehr guter Standard zwischen Genauigkeit und Performance.
- Kleinere `dt` erhöhen Genauigkeit, aber auch Rechenzeit und Speicher.
- Die interne Korrektur hält die Summe stabil, ohne Schwingungen einzubauen.

Performance-Profil

- Zeitkomplexität **O(steps · dims)**, Speicher **O(steps · dims)**.
- Arrays werden einmal erzeugt und zeilenweise befüllt.
- Der Director bündelt häufige Änderungen über `requestAnimationFrame` und reduziert so Rechenlast in UI-Szenarien.

Zusammenspiel mit Director und Bus

Ablauf pro Interaktion

1. Parameteränderung kommt beim Director an und wird normalisiert und gekoppelt.
2. Der Director ruft `engine.run(...)` und erhält `{ series, meta, drift }`.
3. Der Director publiziert `uid:e:model:update`, `uid:e:sim:data` und `uid:e:engine:status`.
4. Chart, KPI und Vektorrad reagieren auf die Daten. Pointerbewegungen lösen **keine** Neuberechnung aus.

Fehlerfall

- Wenn die Invarianten verletzt wirken, sendet der Director `uid:e:error` und stellt Kontext bereit.

Typische Fehlerbilder und Gegenmittel

- **Negative Bestände** durch zu große Schritte. Gegenmittel `dt` verkleinern oder `rk4` nutzen.
- **Drift** bei langen Läufen. Gegenmittel integrierte Massenkorrektur ist aktiv, zusätzlich `dt` prüfen.
- **Unrealistische Peaks** bei extremen Parametern. Parametergrenzen im Katalog beachten und Kopplungen nutzen.

Erweiterbarkeit

Neues Modell hinzufügen

```
const MODELS = {
  ...,
  NEW: {
    dims: ['S','X','I','R'],
    init: (p) => [Math.max(0, p.N - p.I0), 0, p.I0, 0],
    deriv: (p, y) => { /* d/dt(y) als Array */ }
  }
}
```

- **dims** definiert Reihenfolge und erzeugt die Series-Keys.
- **init** liefert Startvektor aus Parametern.
- **deriv(p,y)** gibt pro Kompartiment die Zeitableitung zurück.

Neuer Integrator

```
function stepAB2(f,p,y,h){ /* Adams-Bashforth 2 */ }
const STEPPERS = { ... , ab2: stepAB2 };
```

Mini-Rezepte

- **Impfrate aktivieren** im SIRV: `nu > 0` verschiebt Masse aus `S` nach `V` und reduziert so die effektive Ansteckungsfläche.
- **Maßnahmen** als Prozentwert `measures` setzen und damit `β_eff` verringern, ohne Kopplungen aufzubrechen.
- **Latenz** im SEIR über `L` steuern, `σ = 1/L` wird automatisch abgeleitet.

Quick-Map Dateien

- `12-2_base/engine.js` Rechenkern, Integratoren, Modelle, Drift-Korrektur
- `12-2_base/schema.js` Katalog, Normalisierung, Kopplungen
- `12-2_base/uid.js` Director, Planung und Bus-Publikation
- `12-2_base/bus.js` Event-Transport