Simulator of chemical reactions with external events

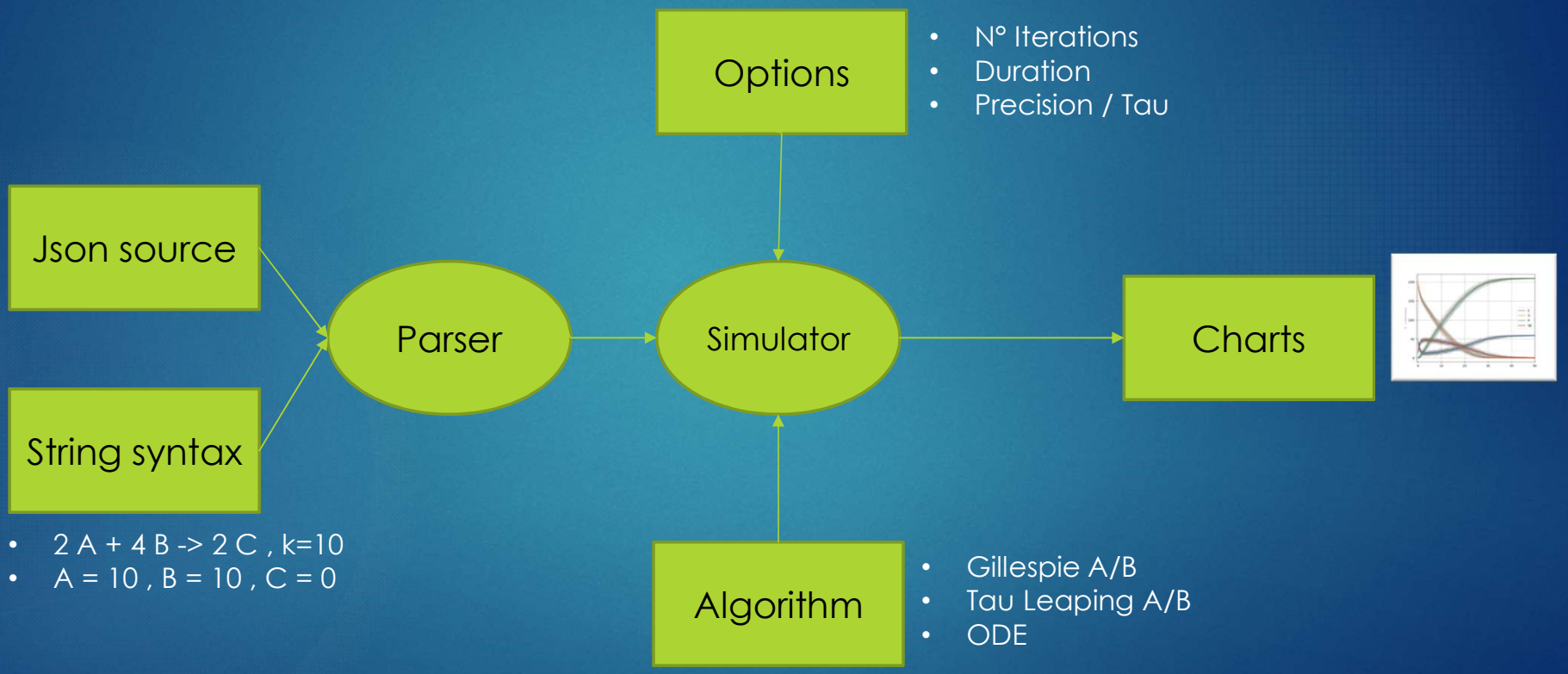
COMPUTATIONAL MODEL FOR COMPLEX SYSTEM COURSE 22/23

M.SC. COMPUTER SCIENCE, ARTIFICIAL INTELLIGENCE

UNIVERSITY OF PISA

Iommi Andrea - 578212

Pipeline


Options
N° Iterations
Duration
Precision / Tau
Json source
String syntax
Parser
Simulator
Charts
2 A + 4 B -> 2 C , k=10
A = 10 , B = 10 , C = 0
Algorithm
Gillespie A/B
Tau Leaping A/B
ODE

Example of json source

```
{
  "reactions": [
    {
      "name": "Reaction 1",
      "reactants": [{"l": "1","molecule": "S"},{"l": "1","molecule": "E"}],
      "products": [{"l": "1","molecule": "SE"}],
      "kinetic" : 0.006
    },
    {
      "name": "Reaction 2",
      "reactants": [{"l": "1","molecule": "SE"}],
      "products": [{"l": "1","molecule": "S"},{"l": "1","molecule": "E"}],
      "kinetic" : 0.005
    },
    {
      "name": "Reaction 3",
      "reactants": [{"l": "1","molecule": "SE"}],
      "products": [{"l": "1","molecule": "P"},{"l": "1","molecule": "E"}],
      "kinetic" : 0.2
    }
  ],
  "initial_state": [
    {"molecule": "E" ,"qnt": 50},
    {"molecule": "S" ,"qnt": 200},
    {"molecule": "P" ,"qnt": 0},
    {"molecule": "SE","qnt": 10}
  ],
  "events": [
    {"time": 20, "molecule": "S","qnt":  10},
    {"time": 30, "molecule": "P","qnt": -10}
  ]
}
```

Algorithms

- Stochastic simulation methods
  - **Gillespie**
    - Implementation A with Python dictionary
    - Implementation B with numpy arrays
  - **Tau leaping**
    - Implementation A with Python dictionary
    - Implementation B with numpy arrays
- Exact methods
  - **Differential Equations**

Stochastic simulation - Gillespie

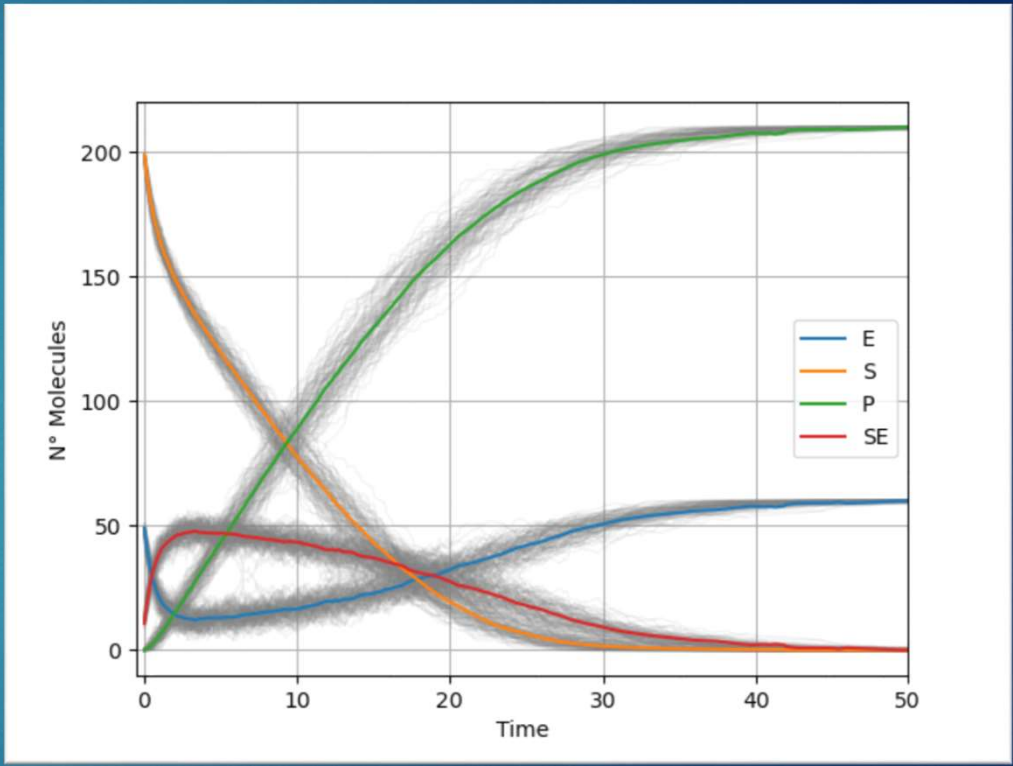
- As example we have taken into account the following chemical reactions.
- It was performed 100 simulation and it was taken a sort of mean of all them.

$$\begin{cases} S + E \rightarrow SE, & k = 0.006 \\ SE \rightarrow S + E, & k = 0.005 \\ SE \rightarrow P + E, & k = 0.200 \end{cases}$$

E = 50 ; S = 200 ; P = 0 ; SE = 10

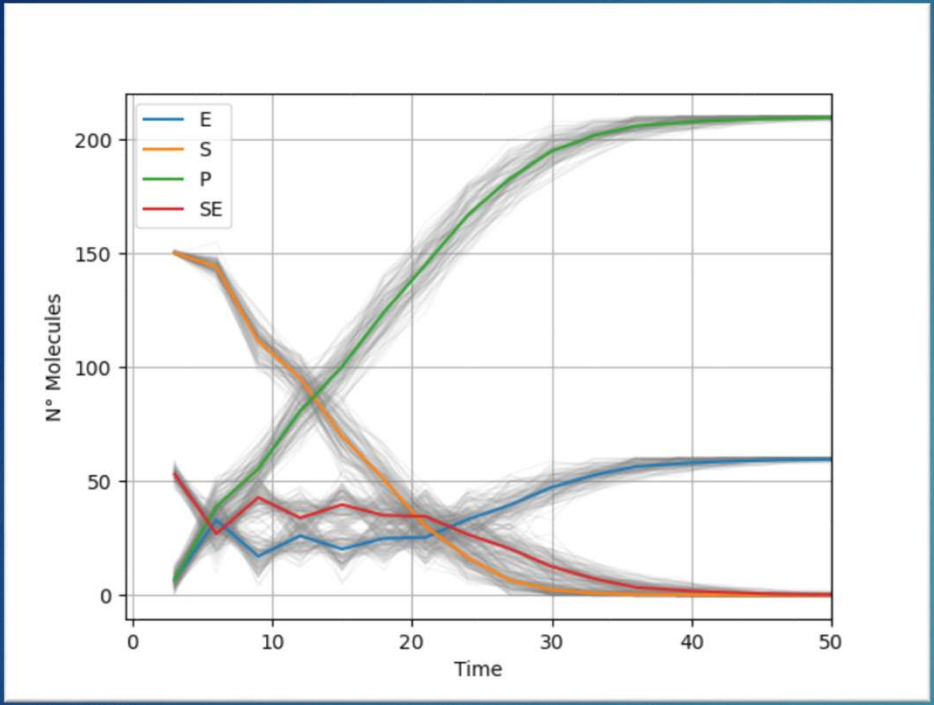
- Gillespie is the most famous algorithm for the stochastic simulation.

- In the next slide we will see a specific and approximate variant.



1.4118 *s*

Stochastic simulation - Tau Leaping

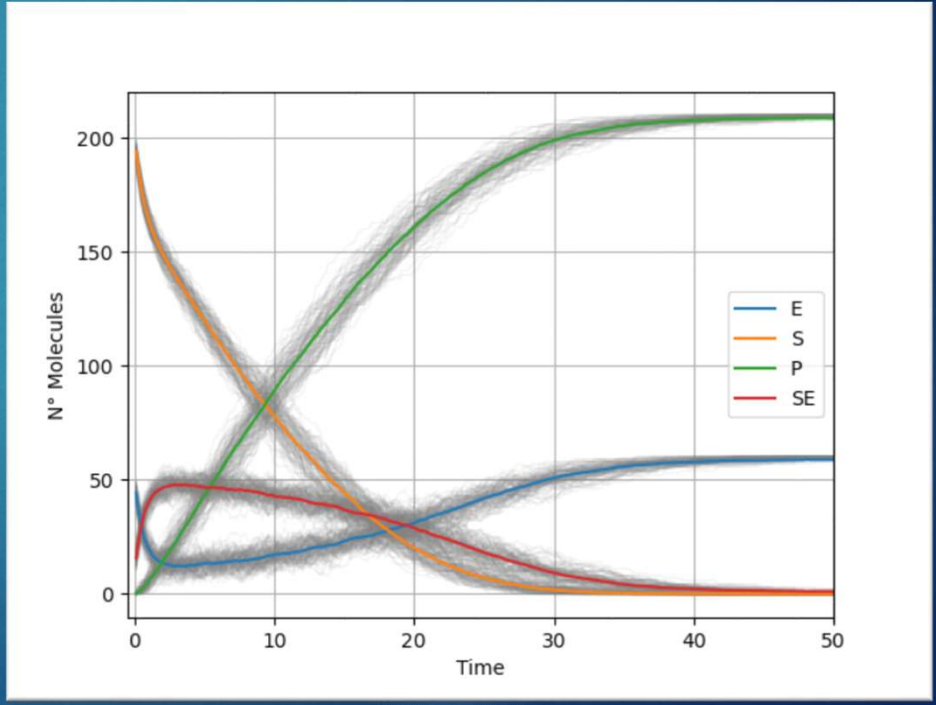


7.10e-2 *s with tau =* ***3***

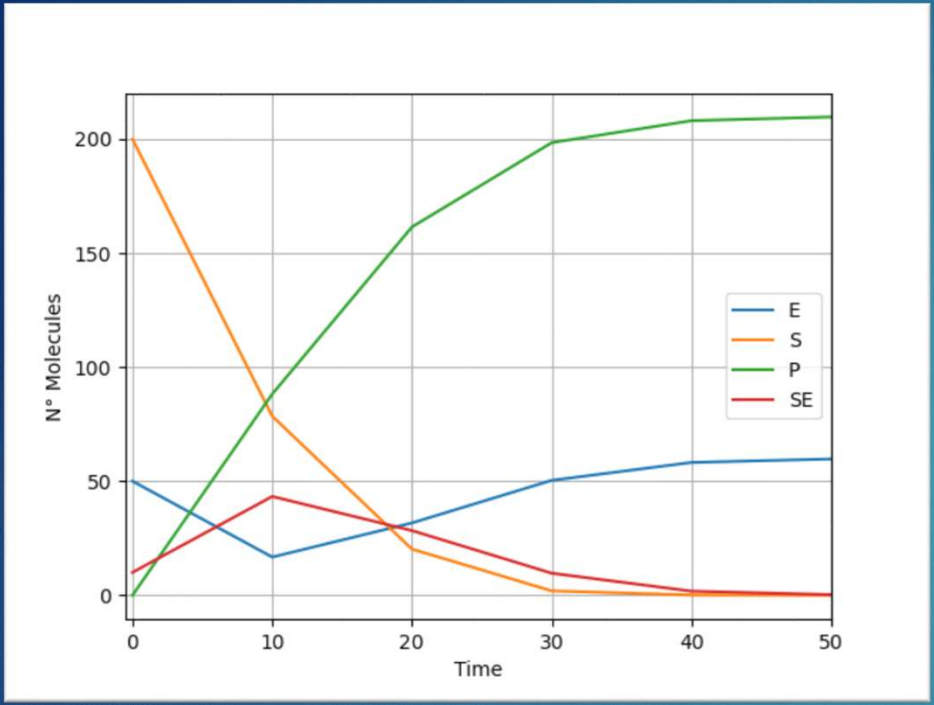
- Instead decreasing the step, the execution becomes more smooth but slower.

- As we can see, with **high tau** step, the execution is extremely fast but not accurate.

1.285 *s with tau =* ***0.1***



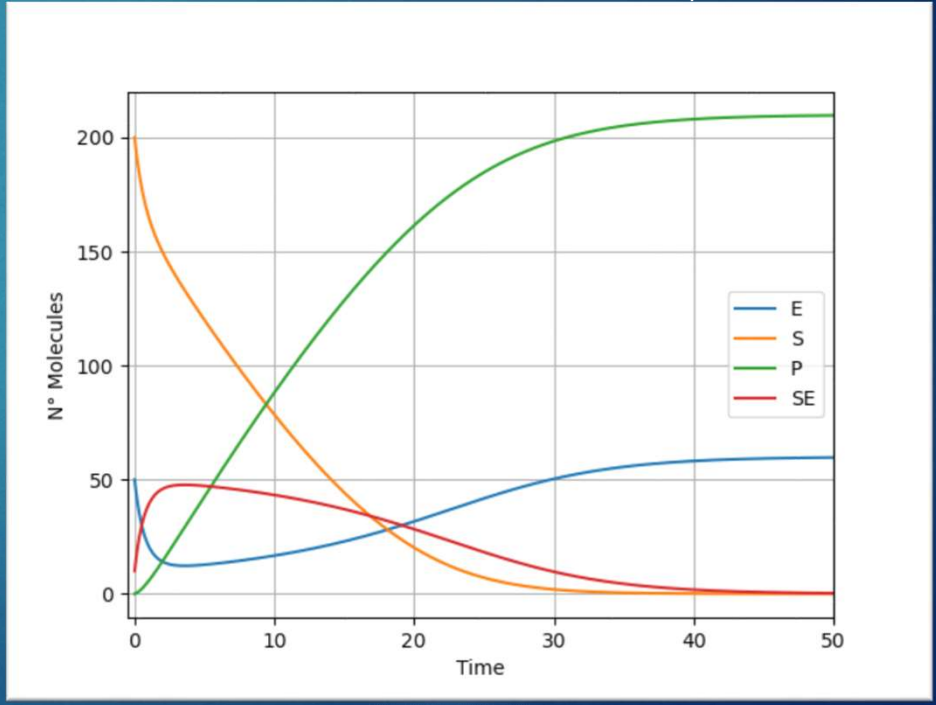
Exact methods - Differential equations



7.0e-3 *s with precision =* ***8***

- The ODE prove an exact solution, the interval's precision does **not impact** on performance.

8.0e-3 *s with precision =* ***0.1***



Implementations : Dicts vs arrays

Andrea *

```
def step(self) -> float:
    # perform propensities (instantaneous rate of each reaction)
    n = len(self.reactions)
    a, a_0 = zeros(n), 0
    # calculate the propensities
    for idx, react in enumerate(self.reactions):
        _a = react.kinetic
        for r, l in react.reactants.items():
            # Oss if the molecule quantities is 0 then _a become 0
            _a *= binom(self.state[r], l)
        a[idx] = _a
        a_0 += _a
    # if all propensities are zero means that there are
    # no reaction to execute
    if a_0 == 0:
        return -1
    a /= a_0
    # time event occurs
    dt = math.log(1 / random.uniform(0, 1)) / a_0
    # select the reaction to perform
    react = self.reactions[choice(arange(0, n), p=a)]
    # update the state with the reaction chosen
    for r, l in react.reactants.items():
        self.state[r] -= l
    for r, l in react.products.items():
        self.state[r] += l
    return dt  # tau
```

Andrea *

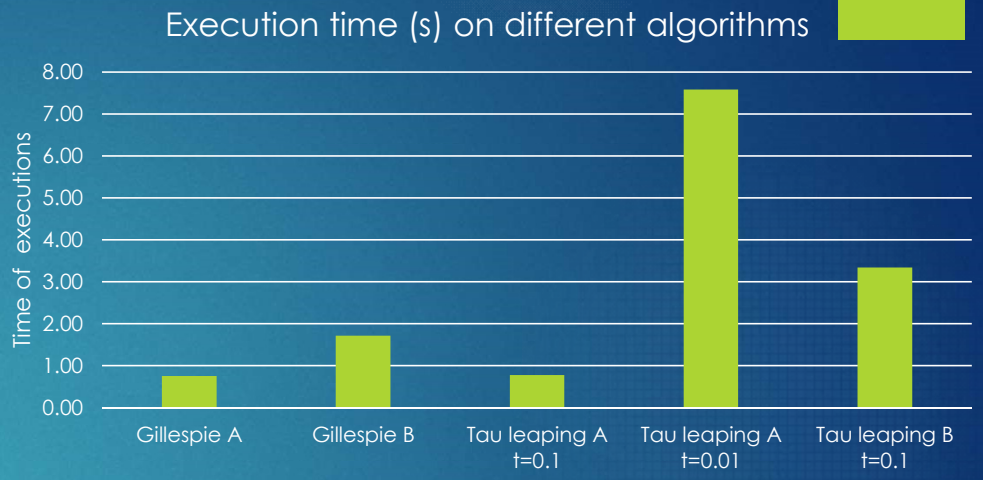
```
def step(self) -> float:
    n = self.kinetic.shape[0]
    # perform propensities (instantaneous rate of each reaction)
    a = vectorize(binom)(self.state, self.reactants) \
        .prod(axis=1) * self.kinetic

    a_0 = a.sum()
    # if all propensities are zero means that there are
    # no reaction to execute
    if a_0 == 0:
        return -1
    a /= a_0
    # time event occurs
    dt = math.log(1 / random.uniform(0, 1)) / a_0
    # update num of molecules based on reaction occurred
    c = choice(arange(0, n), p=a)
    self.state += self.products[c, :] - self.reactants[c, :]
    return dt  # tau
```

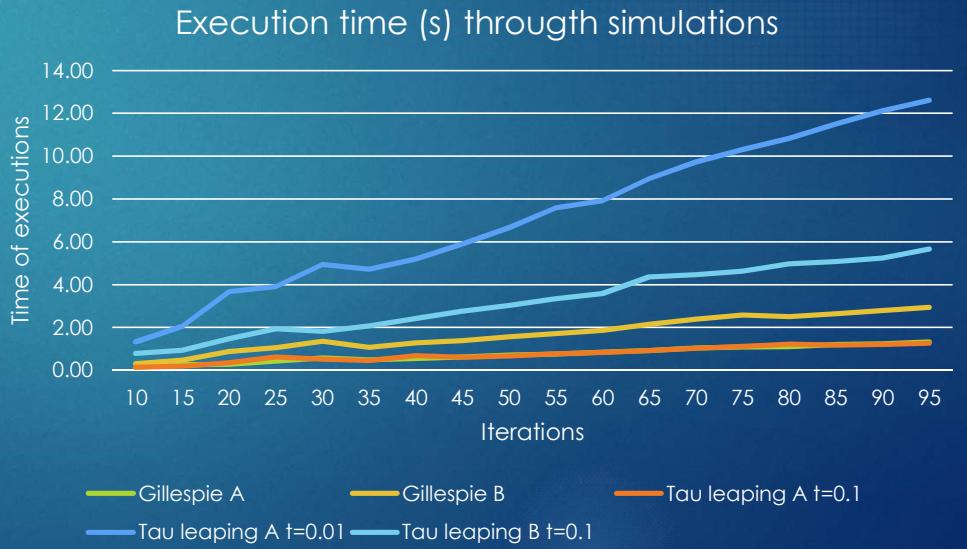
- On the left there is a dictionary-based implementation, on the right a matrix-based.
- We can see that the first implementation is more intuitive but less compact respect to the second one.

Performances among stochastic algorithms

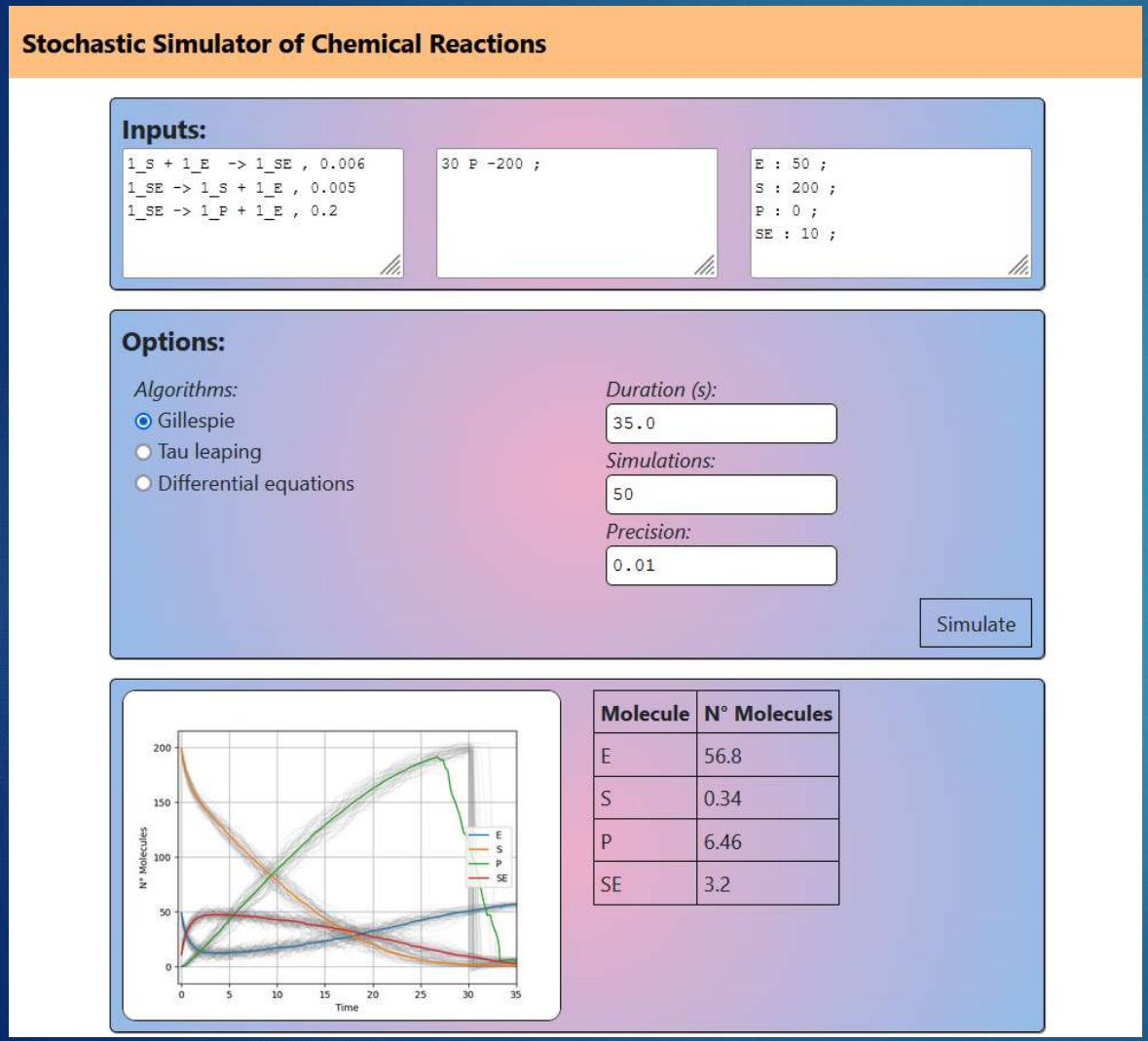
- The tests were performed with Ryzen 7 4800H as CPU and Win10.
- In the first chart all the executions was performed with **fixed** duration and iterations. (itr =55 , end_time=50.0)
- As expected the **Tau leaping** with high Tau achieves the best speed.
- We can notice that the matrix based performs worse than dictionary based.

(*This could be explained by the fact that we have taken into account only very small reactions and a low number of molecules, the matrix based can be worth were are involved thousand of molecules or reactions, because it exploits a vectorized functions and others mechanisms that the dictionary do not have*).





Graphical web interface with Flask



- On the left we have a naïve web **graphical interface**.
- **In input** we can insert the reactions, events and the initial state with a simple syntax.
- Then the next step is to choose which kind of algorithm to simulate and the option like **duration** of simulation* , **the number of simulation** (to compute the average) and the **precision** (used for tau leaping and ode)
- **As result** we have the chart and the final state.

**it doesn't represent the time paid to execute the simulation, but the time of experiment*

References

- Source code:
  - https://github.com/jacons/Chemical-Reactions-Simulator
- Algorithms:
  - https://elearning.di.unipi.it/course/view.php?id=292
  - Slides -> 05-StochasticSimulationChemicalReactions.pdf