Our Case Study

Our Case Study

As our next case study we will consider an epidemic control problem

- Let's assume we are at early stages of an epidemic
- ...And we want to do our best to control while we wait for a cure/vaccine

Epidemic Control as Optimization

Technically, this is an optimization problem

- We need to decide which actions to take
- ...Subject to a variety of constraints (e.g. socio economical impact)
- ...So that the total number of infected is minimized

But how do we evaluate the impact of our actions?

Epidemic Control as Optimization

Technically, this is an optimization problem

- We need to decide which actions to take
- ...Subject to a variety of constraints (e.g. socio economical impact)
- ...So that the total number of infected is minimized

But how do we evaluate the impact of our actions?

Epidemical dynamics can be simulated

- We can use differential equations
- We can use multi-agent models
- We can use networks to account for connections

Epidemic Control as Optimization

Technically, this is an optimization problem

- We need to decide which actions to take
- ...Subject to a variety of constraints (e.g. socio economical impact)
- ...So that the number of infected is minimized

But how do we evaluate the impact of our actions?

However, using such simulator in optimization is difficult

- They are defienned via rules and/or equations
- ...But using those in a declarative optimization approach is prohibitive
- Black-box optimization is an option, but cannot deal easily with constraints

Empirical Model Learning

We will tackle this problem via Empirial Model Learning

The key idea in EML is to make ML models part of optimization models

- We have learned how to inject a constraint model in ML...
- ...And now we will see how to inject ML into a constraint model

EML was designed to enable optimization over complex systems, e.g.

- Traffic optimization
- Thermal aware (computational) job scheduling
- Design of incentive schemes
- ...

At its simplest, the approach requires to

- Learn a ML model in any usual way
- Find a way to encode/embed the model in a given optimization technology

Compartmental Models for Epidemics

For our epidemics we will rely on a SIR model

SIR models are a type of compartmental model

- The population is divided into three groups (compartments)
- ...I.e. Susceptibles, Infected, Recovered

The classical SIR model is dynamic system

- The size of the three groups evolves over time
- According to an Ordinary Differemtial Equation (ODE)

An ODE is a differential equation in the form:

$$\dot{y} = f(y, t)$$

- y is a (vector) variable representing the system state
- $f(y, t)$ defines the gradient of the state

Compartmental Models for Epidemics

In the case of the SIR model, we have:

$$\dot{S} = -\beta \frac{1}{N} SI$$

$$\dot{I} = \beta \frac{1}{N} SI - \gamma I$$

$$\dot{R} = \gamma I$$

Where:

- S, I, R refer to the size of each component
- N is the population size (i.e. $N = S + I + R$
- ...β is the infection rate and γ the recovery rate
- ...And the ratio $R_0 = \beta/\gamma$ is called basic reproductive number

Compartmental Models for Epidemics

In the case of the SIR model, we have:

$$\dot{S} = -\beta \frac{1}{N} SI$$

$$\dot{I} = \beta \frac{1}{N} SI - \gamma I$$

$$\dot{R} = \gamma I$$

We have that:

- S decreases proportionally to the product SI
- I grows by the same rate, and decreases proportionally to its size I
- R grows proportionally to I

Individiduals "flow" from S to I, and then to R

Solution Methods for ODEs

Solving an ODE can be thought of as running a simulation

The simples solution approach is called Euler's method

- $y(0) = \text{initial system state}, t(0) = 0$
- for $i = 1..n_{steps}$
  - $t(i) = t(i-1) + h$
  - $y(i) = y(i-1) + hf\,(y(i-1), t(i-1))$

Intuitively:

- We start from an initial state $y(0)$
- We make linear updates to the state, with step h
- ...Using $f(y, t)$ to compute the gradient
- We keep track of passing time in t

Solving Our SIR

First, we need to define a function to compute our gradient

I.e. we simply need to compute the SIR formulas

```
def SIR(y, beta, gamma):
    # Unpack the state
    S, I, R = y
    N = sum([S, I, R])
    # Compute partial derivatives
    dS = -beta * S * I / N
    dI = beta * S * I / N - gamma * I
    dR = gamma * I
    # Return gradient
    return np.array([dS, dI, dR])
```

- Rather than passing N as a parameter
- ...Here we compute it on the fly

Solving Our SIR

Then we just need to call an ODE integration function

We will use `odeint` from scipy

```
odeint(f, y0, t, ...)
```

- `y0` is the initial state
- `t` is vector of time points
  - The function allows for variable time steps
  - ...Defined via the differences `t[1:] - t[:-1]`
- `f` is the gradient computation function
  - ...And it should implement the interface `f(y, t)`
  - Alternative interfaces are possible (but we won't cover them)

Solving Our SIR

Our simulation code is in `simulate_SIR`

```
def simulate_SIR(S0, I0, R0, beta, gamma, tmax, steps_per_day=1):
    # Build initial state
    y0 = np.array([S0, I0, R0])
    # Wrapper
    nabla = lambda y, t: SIR(y, beta, gamma)
    # Solve
    t = np.linspace(0, tmax, steps_per_day * tmax)
    Y = odeint(nabla, y0, t)
    # Wrap as dataframe and return
    ...
```

- We use a `lambda` function to obtain the expected interface
- Each time unit corresponds to one day
- ...And we control the number of steps per day with `steps_per_day`

Solving Our SIR

Next, we define test values for all the parameters

In [2]:
```
S0, I0, R0 = 0.99, 0.01, 0.0
beta, gamma = 0.1, 1/14
tmax = 365
```

- We consider a normalized population ($N = 1$)
- Initially, 1% of the population is infected
- γ is the inverse of the average recovery time (14 days)
- We simulate for one year

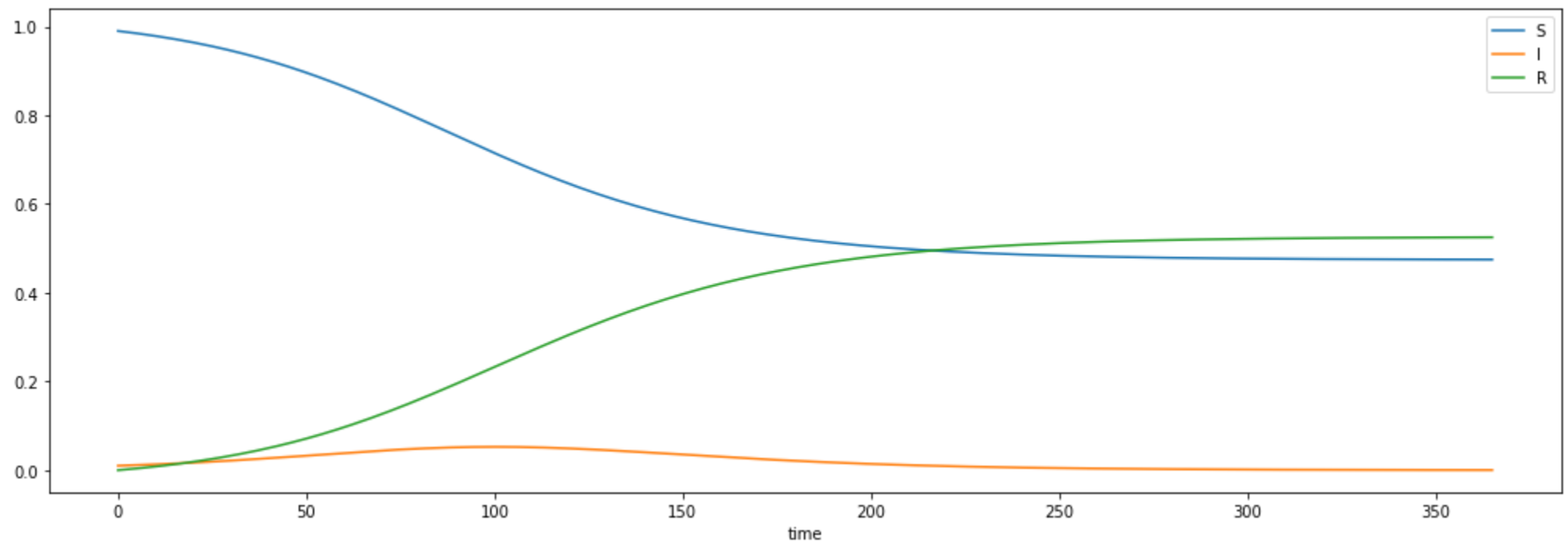
The value of R_0 determines whether we have a proper epidemic behavior

- If $R_0 > 1$ infections grow before falling, otherwise they only decrease
- We have $R_0 = \beta/\gamma = 1.4$, i.e. a true epidemic behavior

Solving Our SIR

Let's plot the dynamics for one year

In [3]:
```
data = util.simulate_SIR(S0, I0, R0, beta, gamma, tmax=tmax, steps_per_day=1)
util.plot_df_cols(data, figsize=figsize)
```



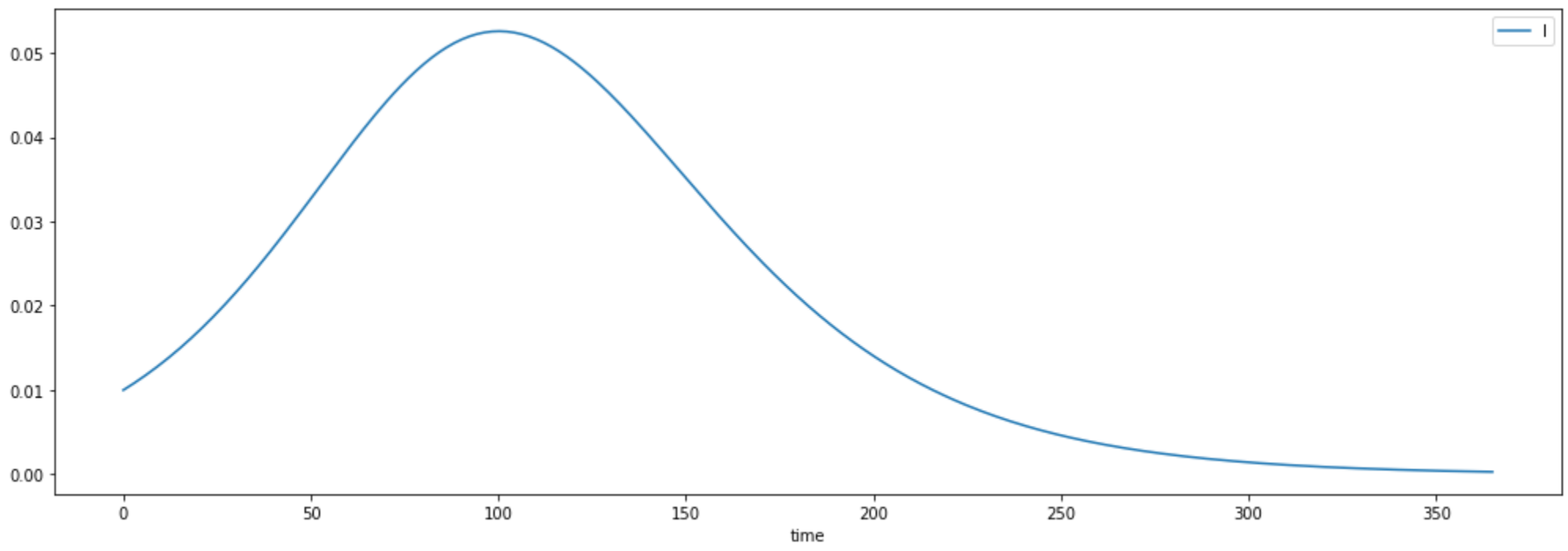
- The S compartment monotonically decreases
- The R compartment monotonically increases

Solving Our SIR

Let's focus on the infected curve

The number of infected grows, before decreasing again

In [4]:
```
util.plot_df_cols(data[['I']], figsize=figsize)
```



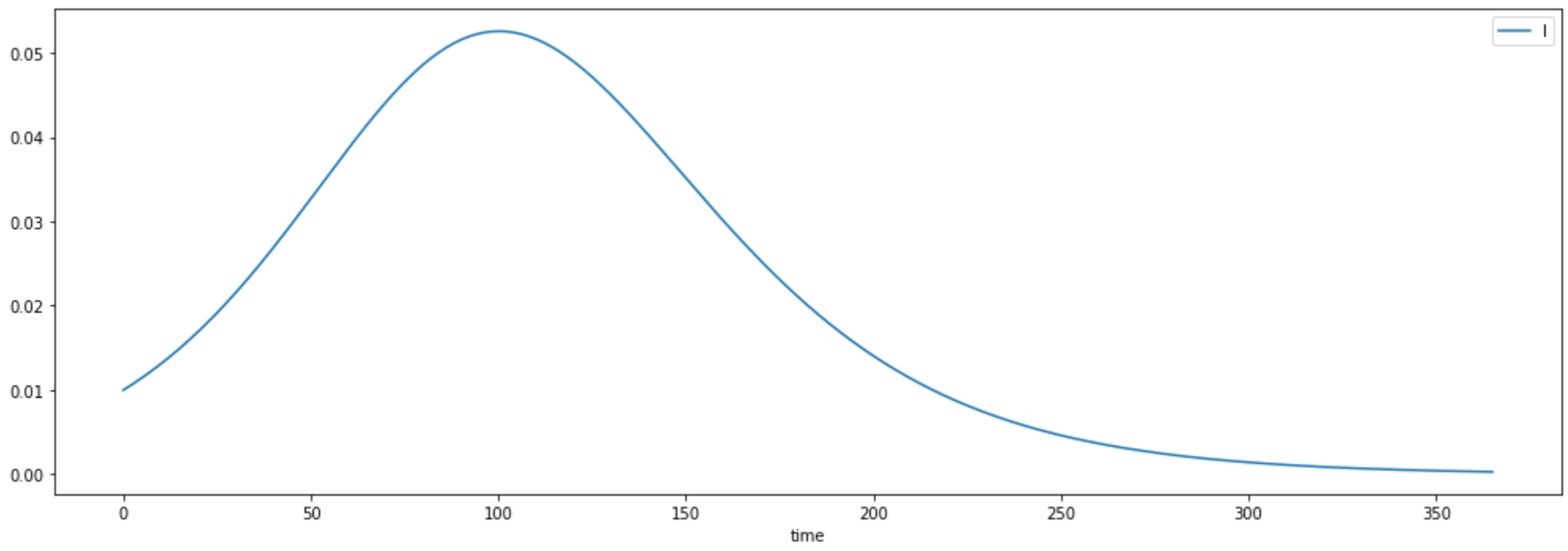
- In a true epidemics, the behavior will be more complicated
- ...In particular, we will typically have multiple waves

Solving Our SIR

Let's focus on the infected curve

The number of infected grows, before decreasing again

In [5]:
```
util.plot_df_cols(data[['I']], figsize=figsize)
```



- However, SIR models are still a good local descriptor
- ...Which is why the R_t values is routinely monitored