```julia
begin
    # numerical libraries
    using Expokit, PROPACK, Arpack, SparseArrays
    # output and plotting
    using ProgressLogging, JLD, CairoMakie
    # modelling and statistics
    using Catalyst, JumpProcesses, StatsBase, DifferentialEquations
    using Interpolations
    # importing local fsp package
    using Revise
    local_mod = include("../src/DiscStochSim.jl")
    using .local_mod.DiscStochSim
end
```

```
Replacing docs for `Main.var"workspace#4".DiscStochSim.FindLowestValuesPercent :
: Union{Tuple{T}, Tuple{Vector{T}, Number}} where T` in module `Main.var"workspa
ce#4".DiscStochSim`
```

rn =

$$\mathrm{S} + \mathrm{E} \underset{\mathrm{kD}}{\stackrel{\mathrm{kB}}{\rightleftharpoons}} \mathrm{SE}$$

$$\mathrm{SE} \xrightarrow{\mathrm{kP}} \mathrm{P} + \mathrm{E}$$

```julia
rn = @reaction_network begin
    kB, S + E --> SE
    kD, SE --> S + E
    kP, SE --> P + E
end
```

```julia
model = DiscreteStochasticSystem(rn);
```

def_params = 0.0:0.01:200.0

```julia
def_params = begin
    # reaction rates
    rates = [0.01, 0.1, 0.1];
    # boundary
    bounds = (0, 60) #(lower limit, upper limit)
    boundary_condition(x) = RectLatticeBoundaryCondition(x, bounds);
    # time interval and initial values
    δt = 0.01
    T = 0:δt:200
end
```

```
init_fsp_vars = begin
    global U₀ = CartesianIndex(50, 10, 1, 1);
    global 𝒮₀ = Set([U₀])
    global 𝒮₀ = expand!(𝒮₀, model, rates, 0.0, boundary_condition, 1);
    # initial probability vector (only for active states)
    global p₀ = zeros(𝒮₀ |> length)
    global p₀[FindElement(U₀, 𝒮₀)] = 1
end;
```

fsp_sim =

```
fsp_sim = begin

    # copy initial values
    pₜ = copy(p₀)
    𝒮ₜ = copy(𝒮₀)

    # time stepping loop
    iter = 1
    pₜ = p₀

    # variables to store simulation observables
    size_𝒮ₜ = Int.(zeros(length(T))) # system sizes
    ϵₜ = zeros(length(T)) #local truncation err
    sol = []

    @progress for (iter, t) ∈ enumerate(T)

        # expand state space
        global 𝒮ₜ, pₜ = expand!(𝒮ₜ, pₜ, model, rates , t, boundary_condition, 3)
        global size_𝒮ₜ[iter] = length(𝒮ₜ)
        A = MasterEquation(𝒮ₜ, model, rates, boundary_condition, t)

        # solve system and normalize (using expokit)
        global pₜ = expmv(δt, A, pₜ)

        # add add states in sparse arrays
        push!(sol, (𝒮ₜ, pₜ))

        # purge state space
        𝒮ₜ, pₜ = purge!(𝒮ₜ, pₜ, 3.0)
        ϵₜ[iter] = 1.0 - sum(pₜ)
        pₜ ./= sum(pₜ)
    end

end
```

100%

```
20001×4 adjoint(::Matrix{Float64}) with eltype Float64:
 49.956   9.95607  1.04393  1.00002
 49.914   9.91402  1.08598  1.0
 49.8752  9.87521  1.12479  1.0
 49.8393  9.83929  1.16071  1.0
 49.8059  9.80594  1.19406  1.0
 49.7749  9.77491  1.22509  1.0
 49.746   9.74596  1.25404  1.0
  ⋮
 47.1873  7.1873   3.8127   1.00004
 47.1876  7.18768  3.81232  1.00004
 47.1869  7.18964  3.81036  1.00275
 47.187   7.18699  3.81301  1.0
 47.187   7.18695  3.81305  1.0
 47.1873  7.18734  3.81266  1.0
```

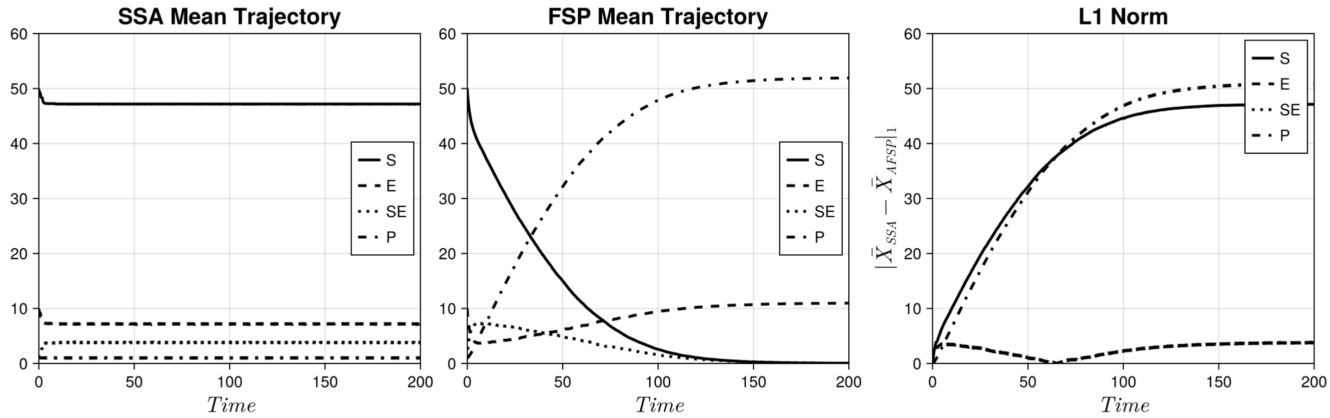
```julia
begin
    sol_mean = map(1:length(T)) do i
        sum(collect.(Tuple.(sol[i][1])) .* sol[i][2])
    end
    fsp_mean=hcat(sol_mean...)'
end
```

```julia
begin
    u0_integers = [:S => 50, :E => 10, :SE => 1, :P => 1]
    tspan = (0., 200.)
    ps = [:kB => 0.01, :kD => 0.1, :kP => 0.1]

    jinput = JumpInputs(rn, u0_integers, tspan, ps)
    jprob = JumpProblem(jinput)
    jump_sol = solve(jprob; seed=1234)

    n_trajs = 1000
    ssa_trajs1=[]
    @progress for i in 1:n_trajs
        push!(ssa_trajs1, solve(jprob, SSAStepper()))
    end;
end
```

```
100%
```

```
(20001-element LinRange{Float64, Int64}:                                        , 20
  0.0, 0.01, 0.02, 0.03, 0.04, 0.05, 0.06, …, 199.96, 199.97, 199.98, 199.99, 200.0   5
```

```julia
uniform_time, ssa_mean = mean_trajectory(ssa_trajs1, 0.0, 200.0, length(T))
```


SSA Mean Trajectory
FSP Mean Trajectory
L1 Norm
Time
S
E
SE
P
$|\bar{X}_{SSA} - \bar{X}_{AFSP}|_1$