Implement Abuduction-Action-Prediction on Lotka-Volterra Using Omega

In this notebook we will use the abduction-action-prediction algorithm to answer the question, "Given a certain number of predators and prey at time t+k, how many predators and prey would there have been if predators stopped eating from time t to t+j, where j < k?". In other words we will be tring to infer the trace of prey and predators using the following equation:
$P(N_{prey}, N_{pred}|Prey_{t+k} = b, Pred_{t+k} = c)$.

In this case we will run Lotka-Volterra with t = 750, j = 100, and k = 250. This means our observational values will be at time 1000.

Packages

In [ ]:

```
## Load Packages
using Omega
using StatsBase
using Random
using Plots
using Distributions
```

Define Omega Functions

In [2]:

```
function get_hazards(rng, n)

    """
    Compute the hazard function given the current states. "spawn_prey" represents
    the event of a prey being born, "prey2pred" represents a predator consuming
    a new prey and consequently spawning a new predator, "pred_dies" represents
    the death of a predator. The function probabilistically selects one of these
    based on their weights. NOTE this version uses distinct values for rates,
    rather than random varaibles.

    args:
        rng(): julia base random number generator, do not need to explicitly pass
        n(int): An index to the current step in prey and pred lists. Used to pull
                most recent values for calculations
    """

    ecology = Dict("prey" => prey_list[n](rng), "pred" => pred_list[n](rng))
```

```
    hazards = Dict(
        "spawn_prey" => theta["spawn_prey"] * ecology["prey"],
        "prey2pred" => theta["prey2pred"] * ecology["prey"] * ecology["pred"],
        "pred_dies" => theta["pred_dies"] * ecology["pred"]
        )

    vals = collect(values(hazards))
    sum_vals = sum(vals)
    prob_vals = vals/sum_vals
    categorical(rng, prob_vals)
end

function one_simulation_prey(rng, n, transitions)

    """
    Simulates one step of gillespie for prey. Takes generated hazards
    adds it to prey and outputs the new value.

    args:
        rng(): julia base random number generator, do not need to explicitly pass
        n(int): An index to the current step in prey and pred lists. Used to pull
                most recent values for calculations
        transitions: Matrix that determines how much prey/pred should change based
                    on selected hazard
    """

    hazard_result = hazards_list[n](rng)
    prey_val = prey_list[n](rng)
    labels = ["spawn_prey", "pred_dies","prey2pred"]
    transition = transitions[labels[hazard_result]]
    new_prey = prey_val + transition[1]

    # Enforce only positive integers
    max(1, new_prey)

end

function one_simulation_pred(rng, n, transitions)

    """
    Simulates one step of gillespie for pred. Takes generated hazards
    adds it to pred and outputs the new value.

    args:
        rng(): julia base random number generator, do not need to explicitly pass
```

```
        n(int): An index to the current step in prey and pred lists. Used to pull
                most recent values for calculations
        transitions: Matrix that determines how much prey/pred should change based
                    on selected hazard
    """

    hazard_result = hazards_list[n](rng)
    pred_val = pred_list[n](rng)
    labels = ["spawn_prey", "pred_dies","prey2pred"]
    transition = transitions[labels[hazard_result]]
    new_pred = pred_val + transition[2]

    # Enforce only positive integers
    max(1, new_pred)

end
```

Out[2]:

```
one_simulation_pred (generic function with 1 method)
```

Groundtruth Counterfactual

In order to better compare the results of the Abuduction-Action-Prediction algorithm, we will first apply the counterfactual to the known ground-truth. In real life this would be impossible, but this is an advantage of simulation.

Initialize Parameters

In [3]:

```
## Transition Matrix
Pre = [[1, 0], [1, 1], [0, 1]]
Post = [[2, 0], [0, 2], [0, 0]]
transition_mat = Post - Pre
transitions = Dict("spawn_prey" => transition_mat[1,],
                    "prey2pred" => transition_mat[2,],
                    "pred_dies" => transition_mat[3,])

# Initial Prey and Pred values
prey_init = normal(60., .001)
pred_init = normal(100., .001)

# Rate Values
spawn_prey = .9
prey2pred = .004
pred_dies = .4

theta =  Dict("spawn_prey" => spawn_prey,
```

```
        "prey2pred" => prey2pred,
        "pred_dies" => pred_dies)
```

Out[3]:

```
Dict{String,Float64} with 3 entries:
  "spawn_prey" => 0.9
  "pred_dies"  => 0.4
  "prey2pred"  => 0.004
```

Build Ground Truth Model

In [4]:

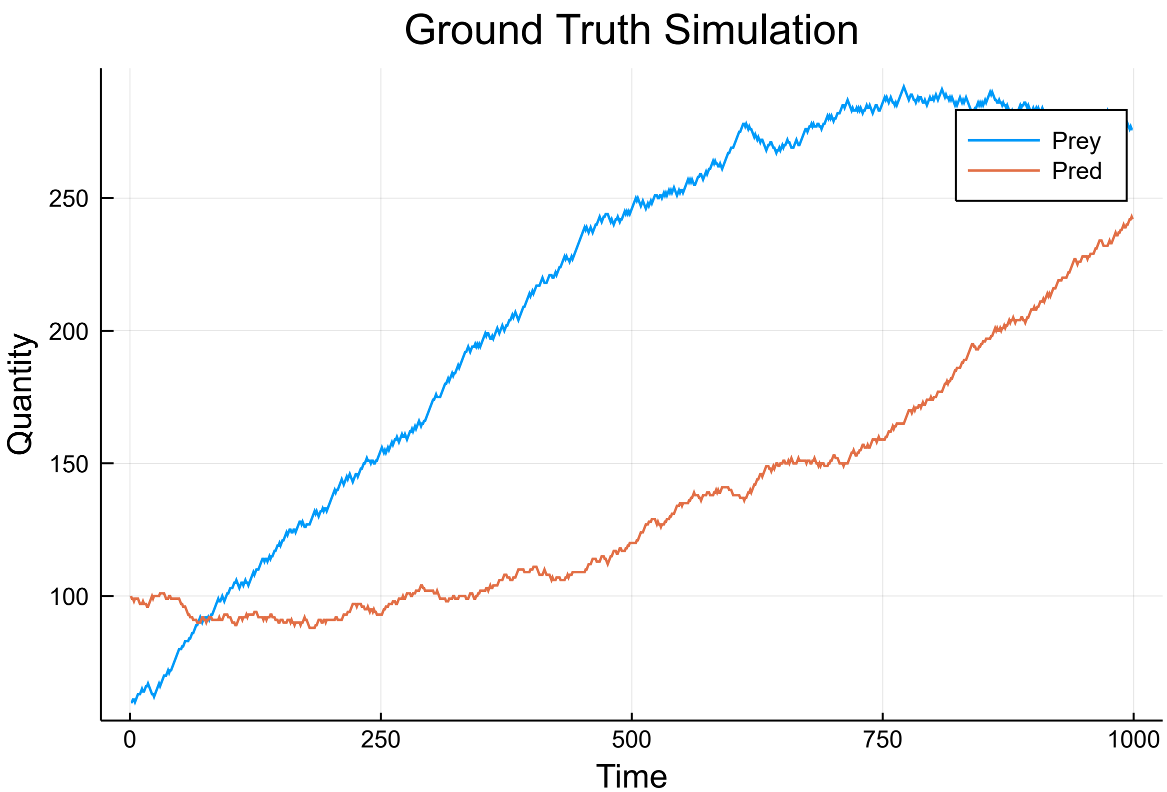
```
## Initialize lists and add starting rand vars
hazards_list = Any[]
prey_list = Any[]
pred_list = Any[]
push!(prey_list, prey_init)
push!(pred_list, pred_init)

## How many time periods to cycle over
N = 1000

## Create a prey/pred/hazard for each time period
for f in 2:N
    last = f - 1
    hazards_temp = ciid(get_hazards, last) # individual step
    prey_temp = ciid(one_simulation_prey, last, transitions) # individual step
    pred_temp = ciid(one_simulation_pred, last, transitions) # individual step
    push!(hazards_list, hazards_temp)
    push!(prey_list, prey_temp)
    push!(pred_list, pred_temp)
end

## Convert lists to single tuple
random_var_tuple = (Tuple(x for x in hazards_list)...,
                Tuple(x for x in prey_list)...,
                Tuple(x for x in pred_list)...)
print()
```

In [6]:

```
## Sample
Random.seed!(1234)
samples = rand(random_var_tuple, 1, alg = RejectionSample)

# extract run results and plot
prey_vals = []
pred_vals = []
for x in 1:(N-1)
```

```
    push!(prey_vals,samples[1][N+x-1])
    push!(pred_vals,samples[1][(N*2)+x-1])
end

plot(hcat(prey_vals,pred_vals),
        title = "Ground Truth Simulation",
        xlabel = "Time",
        ylabel = "Quantity",
        label = ["Prey" "Pred"],
        lw = 1.25)
```

Out[6]:



In [11]:

```
## Prey and Pred at t=1000
prey_vals[999], pred_vals[999]
```

Out[11]:

```
(276.00022971823716, 242.00072833927317)
```

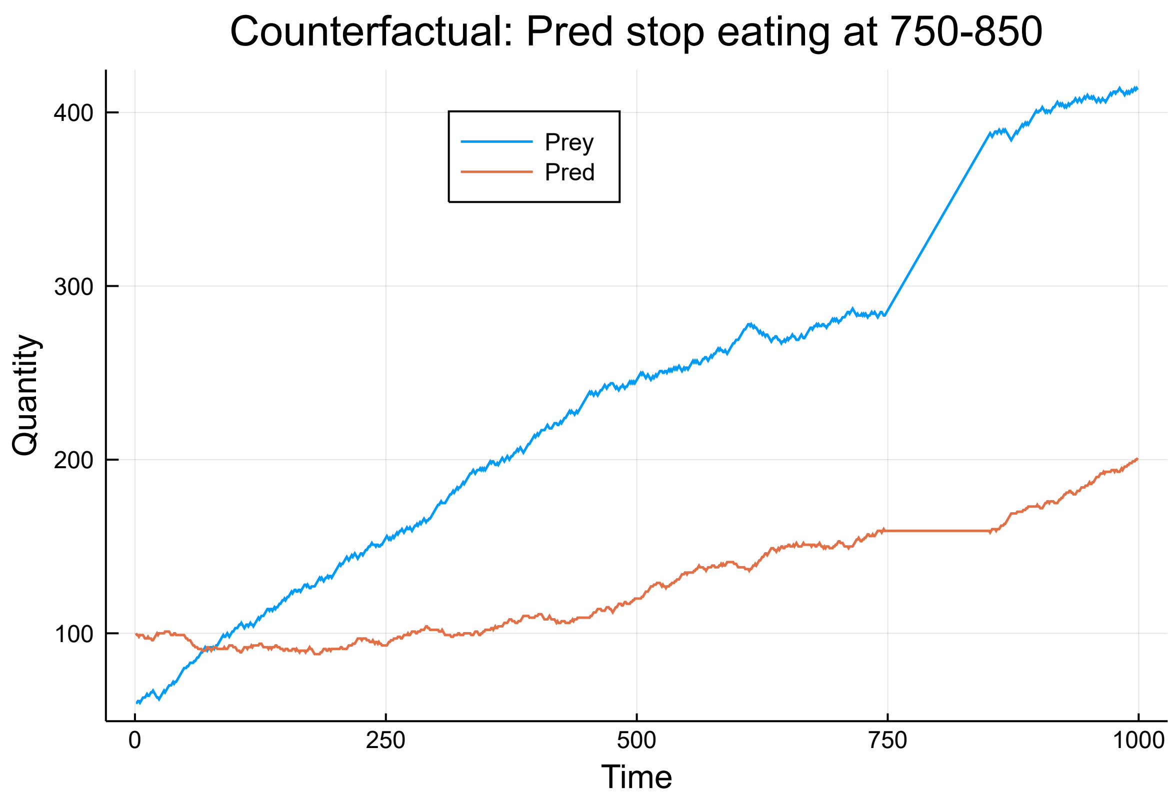
We can see in the ground truth simulation the values of Prey and Pred at time 1000 are 276 and 242 respectively. We will use these values later on in the abuduction step.

Apply Intervention

In [13]:
```
ground_truth_hazards = [x for x in samples[1][1:750]]
ground_truth_prey = [x for x in samples[1][1000:1750]]
ground_truth_pred = [x for x in samples[1][2000:2750]]

for x in 1:100
    push!(ground_truth_hazards, 1)
    push!(ground_truth_prey, ground_truth_prey[length(ground_truth_prey)] + 1)
    push!(ground_truth_pred, ground_truth_pred[length(ground_truth_pred)])
end

## Initialize lists and add starting rand vars
hazards_list = Any[]
prey_list = Any[]
pred_list = Any[]

push!(prey_list, normal(ground_truth_prey[length(ground_truth_prey)], .0001))
push!(pred_list, normal(ground_truth_pred[length(ground_truth_pred)], .0001))

## How many time periods to cycle over
N = 150

## Create a prey/pred/hazard for each time period
for f in 2:N
    last = f - 1
    hazards_temp = ciid(get_hazards, last) # individual step

    ## Condition on prey in this example
    prey_temp = ciid(one_simulation_prey, last, transitions)
    pred_temp = ciid(one_simulation_pred, last, transitions)

    push!(hazards_list, hazards_temp)
    push!(prey_list, prey_temp)
    push!(pred_list, pred_temp)
end

random_var_tuple = (Tuple(x for x in hazards_list)...,
                Tuple(x for x in prey_list)...,
                Tuple(x for x in pred_list)...)

Random.seed!(1234) ## Must initialize seed each time if run line by line
int_samples = rand(random_var_tuple, 1, alg = RejectionSample)

# extract run results and plot
```

```
for x in 2:(N-1)
    push!(ground_truth_prey,int_samples[1][N+x])
    push!(ground_truth_pred,int_samples[1][(N*2)+x])
end

plot(hcat(ground_truth_prey,ground_truth_pred),
        title = "Counterfactual: Pred stop eating at 750-850",
        xlabel = "Time",
        ylabel = "Quantity",
        label = ["Prey" "Pred"],
        lw = 1.25,
        legend = :top)
```

Out[13]:



We can see here that during the times from t to t+j predators stopped eating and prey were allowed to spawn uncontrolled.

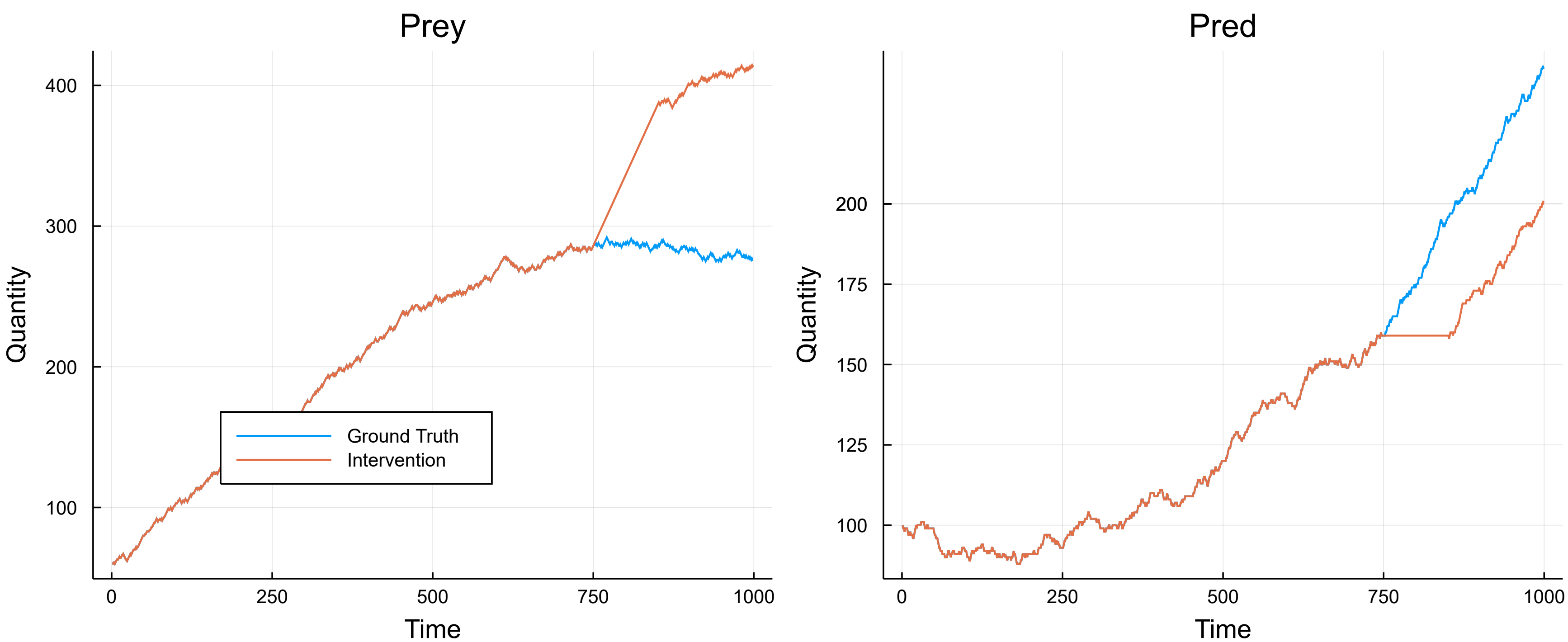
Comparison Plot

In [22]:

```
prey_vals = convert(Array{Float64,1}, prey_vals)
pred_vals = convert(Array{Float64,1}, pred_vals)
```

```
plot1 = plot(hcat(prey_vals, ground_truth_prey),
        title = "Prey",
        xlabel = "Time",
        ylabel = "Quantity",
        label = ["Ground Truth" "Intervention"],
        lw = 1.25,
        legend = false)

plot2 = plot(hcat(pred_vals, ground_truth_pred),
        title = "Pred",
        xlabel = "Time",
        ylabel = "Quantity",
        label = ["Ground Truth" "Intervention"],
        lw = 1.25,
        legend = false)

plot(plot1, plot2,layout = (1, 2), legend = :bottom, size = (1000, 400))
```

Out[22]:



In [23]:
```
## Difference between ground truth and intervention
ground_truth_prey[999] - prey_vals[999], ground_truth_pred[999] - pred_vals[999]
```

Out[23]: (137.00002297182374, -40.999927166072695)

These plots show the ground truth trace vs the intervention. We can see that in the intervention Prey increased by 137 and Pred decreased by 41. We can compare these ground truth counterfactual differences to the abudction-action-prediction we will do in the next step.

Abduction-Action-Prediction

In this section we will infer traces which satisfy the below formula. We will then sample from these traces, apply the counterfactual, and compile the results to a histogram to make a prediction.

$$P(N_{prey}, N_{pred}|\text{Prey}_{T1000} = 276, \text{Pred}_{T1000} = 242)$$

Build Abduction Model

We first build a new model. This is basically the same model as before, but the intial values for Prey and Pred are much wider and sampled uniformly. This is because we do not know the true starting values of our trace, only where they end up. Using this model we can correctly simulate this.

In [25]:

```
## Initialize lists and add starting rand vars
prey_init = uniform(10:150)
pred_init = uniform(10:150)
hazards_list = Any[]
prey_list = Any[]
pred_list = Any[]

push!(prey_list, prey_init)
push!(pred_list, pred_init)

## How many time periods to cycle over
N = 1000

## Create a prey/pred/hazard for each time period
for f in 2:N
    last = f - 1
    hazards_temp = ciid(get_hazards, last) # individual step

    ## Condition on prey in this example
    prey_temp = ciid(one_simulation_prey, last, transitions)
    pred_temp = ciid(one_simulation_pred, last, transitions)

    push!(hazards_list, hazards_temp)
    push!(prey_list, prey_temp)
    push!(pred_list, pred_temp)
end
```

```
random_var_tuple = (Tuple(x for x in hazards_list)...,
                    Tuple(x for x in prey_list)...,
                    Tuple(x for x in pred_list)...)
print()
```

Sample Traces

Below is a large function which will run a simulation and return the traces if they fall within $(\text{Prey}_{T1000} = 276, \text{Pred}_{T1000} = 242)$. Note that a small range around these values was used so that the function could run in a reasonable time.

In [27]:

```
function infer_trace_counterfactuals(N)

    """
    Simulates traces for Prey_T1000 = 276 and Pred_T1000 = 242. Used to infer potential
    traces of prey and pred with only ending values knows

    args:
        N(int): How many traces to simulate
    """

    ## Redefine functions for intervention
    function get_hazards_int(rng, n)

        ecology = Dict("prey" => prey_list_int[n](rng), "pred" => pred_list_int[n](rng))

        hazards = Dict(
            "spawn_prey" => theta["spawn_prey"] * ecology["prey"],
            "prey2pred" => theta["prey2pred"] * ecology["prey"] * ecology["pred"],
            "pred_dies" => theta["pred_dies"] * ecology["pred"]
            )

        vals = collect(values(hazards))
        sum_vals = sum(vals)
        prob_vals = vals/sum_vals
        categorical(rng, prob_vals)
    end

    function one_simulation_prey_int(rng, n, transitions)

        hazard_result = hazards_list_int[n](rng)
        prey_val = prey_list_int[n](rng)
        labels = ["spawn_prey", "pred_dies","prey2pred"]
        transition = transitions[labels[hazard_result]]
```

```
        new_prey = prey_val + transition[1]

        # Enforce only positive integers
        max(1, new_prey)

    end

    function one_simulation_pred_int(rng, n, transitions)

        hazard_result = hazards_list_int[n](rng)
        pred_val = pred_list_int[n](rng)
        labels = ["spawn_prey", "pred_dies","prey2pred"]
        transition = transitions[labels[hazard_result]]
        new_pred = pred_val + transition[2]

        # Enforce only positive integers
        max(1, new_pred)

    end

    global sims = N

    trace_samples = Any[]
    trace_int_samples = Any[]

    ## How many traces do we want to sample
    check = 0
    while check < sims

        ## Sample
        Random.seed!(rand(1:10000000000000))
        samples = rand(random_var_tuple, 1, alg = RejectionSample)

        ## Check if ending values fall into correct range
        if samples[1][1999] <= 278.5 && samples[1][1999] >= 273.5 &&
                samples[1][2999] >= 239.5 && samples[1][2999] <= 244.5

            push!(trace_samples, samples)

            ## Intervene
            ground_truth_hazards = [x for x in samples[1][1:750]]
            ground_truth_prey = [x for x in samples[1][1000:1750]]
            ground_truth_pred = [x for x in samples[1][2000:2750]]

            for x in 1:100
                push!(ground_truth_hazards, 1)
```

```julia
                push!(ground_truth_prey, ground_truth_prey[length(ground_truth_prey)] + 1)
                push!(ground_truth_pred, ground_truth_pred[length(ground_truth_pred)])
            end

            ## Initialize model for counterfactual
            global hazards_list_int = Any[]
            global prey_list_int = Any[]
            global pred_list_int = Any[]

            push!(prey_list_int, normal(ground_truth_prey[length(ground_truth_prey)], .0001))
            push!(pred_list_int, normal(ground_truth_pred[length(ground_truth_pred)], .0001))

            ## How many time periods to cycle over
            N = 150

            ## Create a prey/pred/hazard for each time period
            for f in 2:N
                last = f - 1
                hazards_temp = ciid(get_hazards_int, last) # individual step

                ## Condition on prey in this example
                prey_temp = ciid(one_simulation_prey_int, last, transitions)
                pred_temp = ciid(one_simulation_pred_int, last, transitions)

                push!(hazards_list_int, hazards_temp)
                push!(prey_list_int, prey_temp)
                push!(pred_list_int, pred_temp)
            end

            random_var_tuple_int = (Tuple(x for x in hazards_list_int)...,
                                    Tuple(x for x in prey_list_int)...,
                                    Tuple(x for x in pred_list_int)...)

            int_samples = rand(random_var_tuple_int, 1, alg = RejectionSample)
            push!(trace_int_samples, int_samples)

            ## Add results to output
            check = check + 1
            print(check) ## Print to show progress
        end
    end

    return (trace_samples, trace_int_samples)
end
```

Out[27]: infer_trace_counterfactuals (generic function with 2 methods)

Run simulation. NOTE this takes a long time (~30 minutes) if run at 100 samples. Change input value to something lower to increase speed.

In [30]:
```
trace_sample = infer_trace_counterfactuals(100)
print()
```

```
1234567891011121314151617181920212223242526272829303132333435363738394041424344454647484950515253545556575859606162636465
6667686970717273747576777879808182838485868788899091929394959697989910010
```

Plots of Simulated Traces

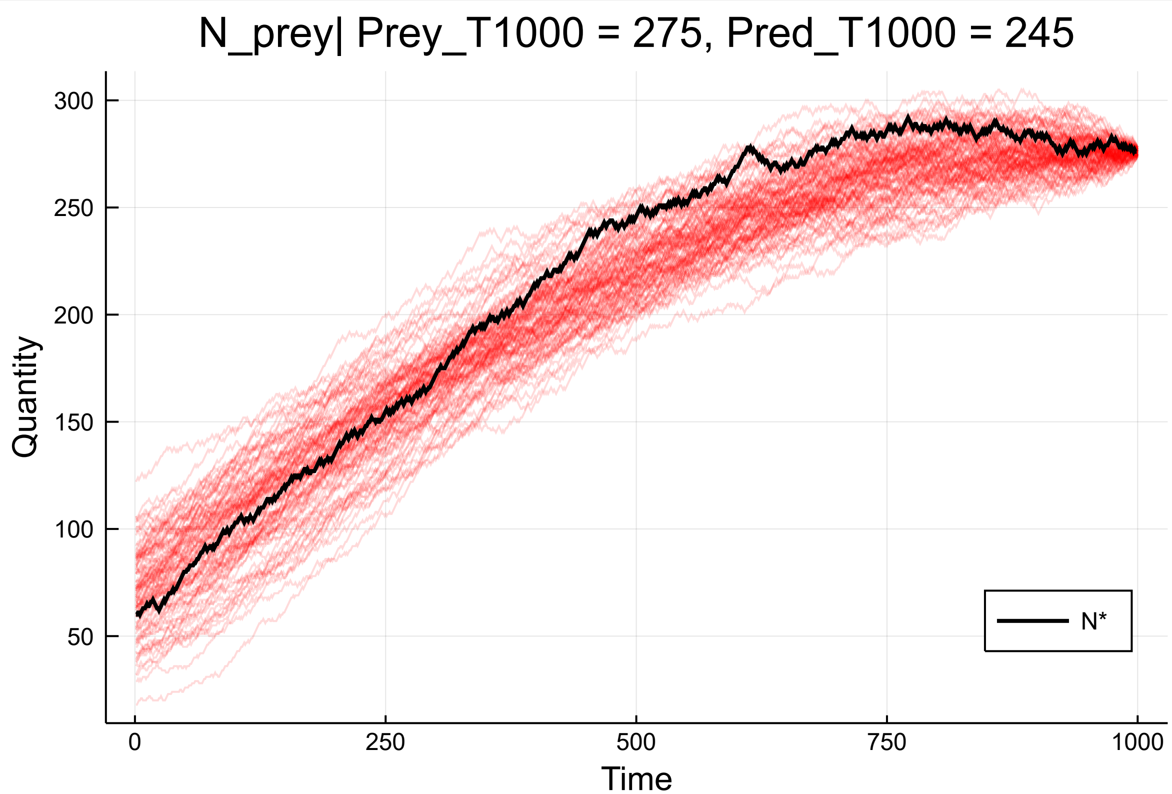
In [31]:
```
## Extract values from simulation
## Line plots
prey_matrix = []
prey_matrix_int = []

pred_matrix = []
pred_matrix_int = []

for idx in 1:length(trace_sample[1])

    push!(prey_matrix, [x for x in trace_sample[1][idx][1][1000:1999]])
    push!(prey_matrix_int, vcat([x for x in trace_sample[1][idx][1][1000:1749]],
                            [trace_sample[1][idx][1][1749] + n for n in 1:100],
                            [x for x in trace_sample[2][idx][1][150:299]]))

    push!(pred_matrix, [x for x in trace_sample[1][idx][1][2000:2999]])
    push!(pred_matrix_int, vcat([x for x in trace_sample[1][idx][1][2000:2749]],
                            [trace_sample[1][idx][1][2749] for n in 1:100],
                            [x for x in trace_sample[2][idx][1][300:449]]))

end

## Histogram
prey_diff = Any[]
pred_diff = Any[]

for idx in 1:length(trace_sample[1])

    prey_temp = trace_sample[2][idx][1][299] - trace_sample[1][idx][1][1999]
    pred_temp = trace_sample[2][idx][1][449] - trace_sample[1][idx][1][2999]

    push!(prey_diff, prey_temp)
    push!(pred_diff, pred_temp)
end
```

Line Plots

In the plots below, the ground truth is represented by the black line.

In [42]:
```
## Prey Traces
plot(title = "N_prey|  Prey_T1000 = 275, Pred_T1000 = 245",
        xlabel = "Time",
        ylabel = "Quantity"
        )
for idx in 1:length(prey_matrix)
    plot!(prey_matrix[idx, :], color = "red", linealpha = .15, legend = false, label = false)
end
plot!(prey_vals, linewidth = 2.0, color = "black", label = "N*", legend = :bottomright)
```

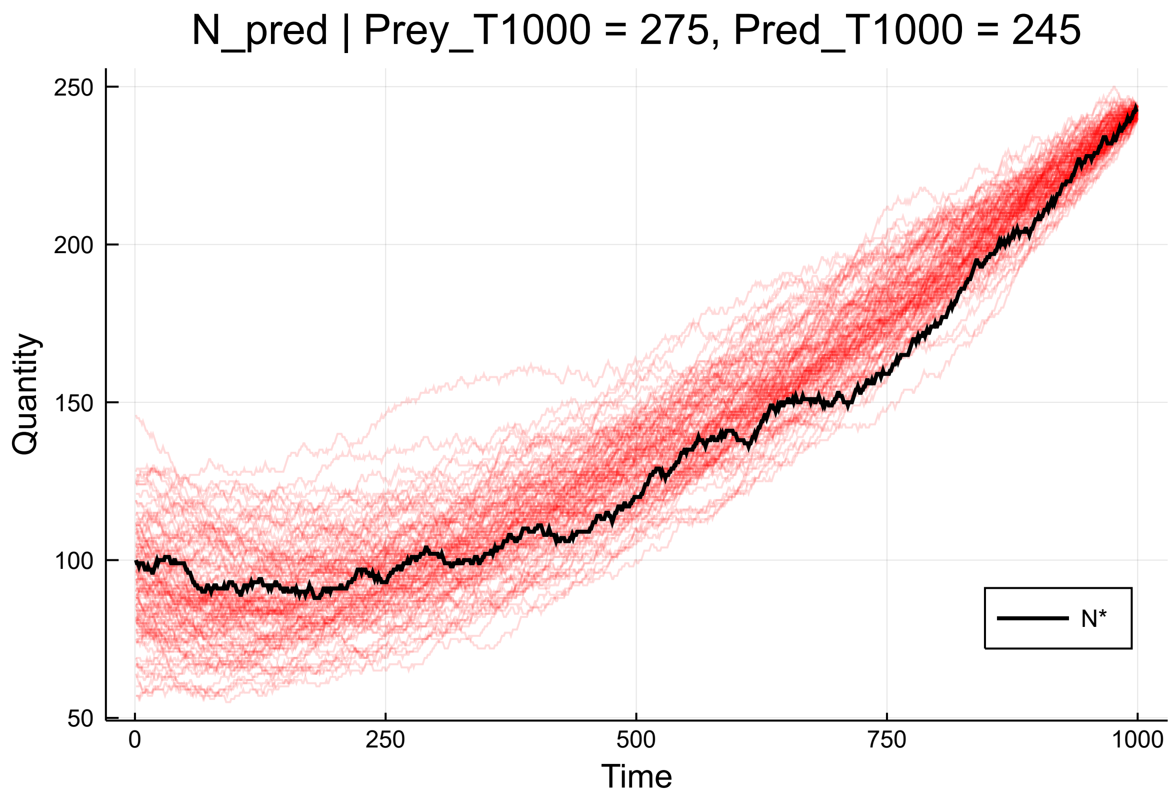
Out[42]:



In [43]:
```
## Pred Traces
plot(title = "N_pred |  Prey_T1000 = 275, Pred_T1000 = 245",
        xlabel = "Time",
        ylabel = "Quantity",
        label = false,
        legend = false
```

```
        )
for idx in 1:length(pred_matrix)
    plot!(pred_matrix[idx, :], color = "red", linealpha = .15, legend = false, label = false)
end
plot!(pred_vals,

    width = 2.0, color = "black", legend = :bottomright, label = "N*")
```

Out[43]:

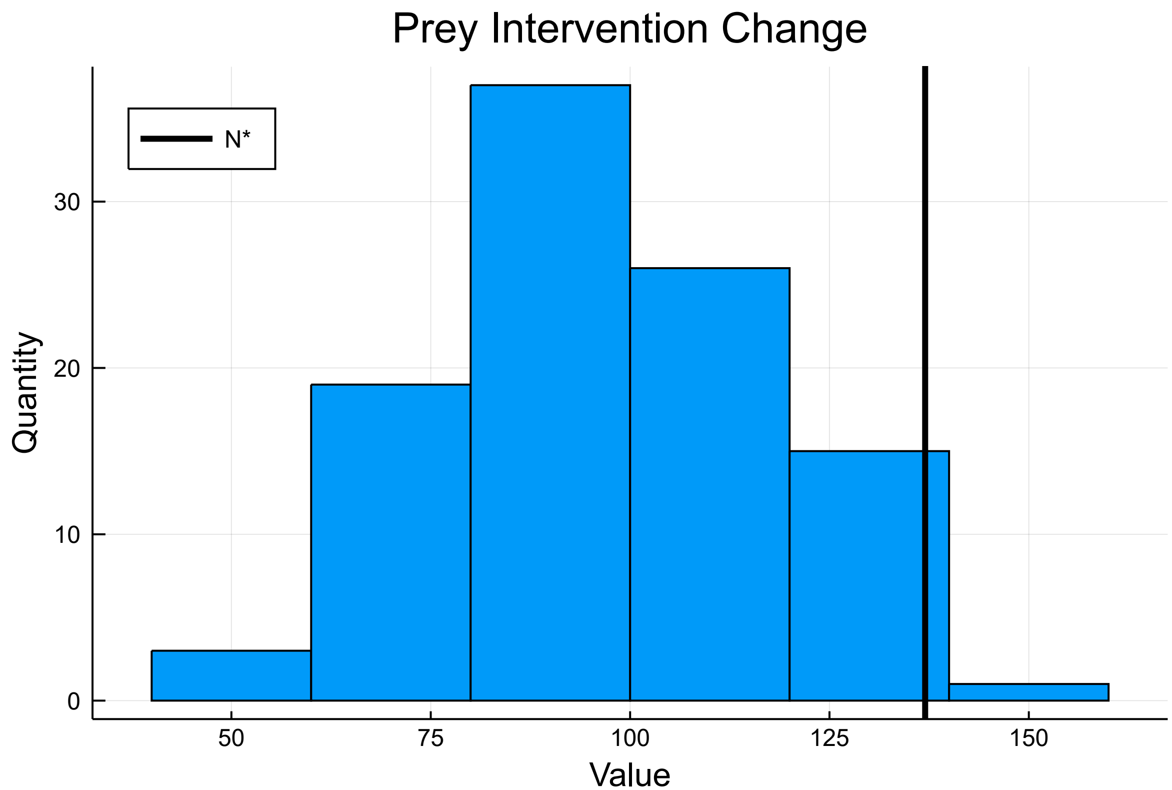


Histograms of Counterfactual Differences

In [81]:

```
plot(prey_diff,
        title = "Prey Intervention Change",
        xlabel = "Value",
        ylabel = "Quantity",
        label = false,
        seriestype = :histogram,
        bins = 8)
plot!([137.], seriestype = :vline, color = "black", lw = 3, label = "N*", legend = :topleft)
```

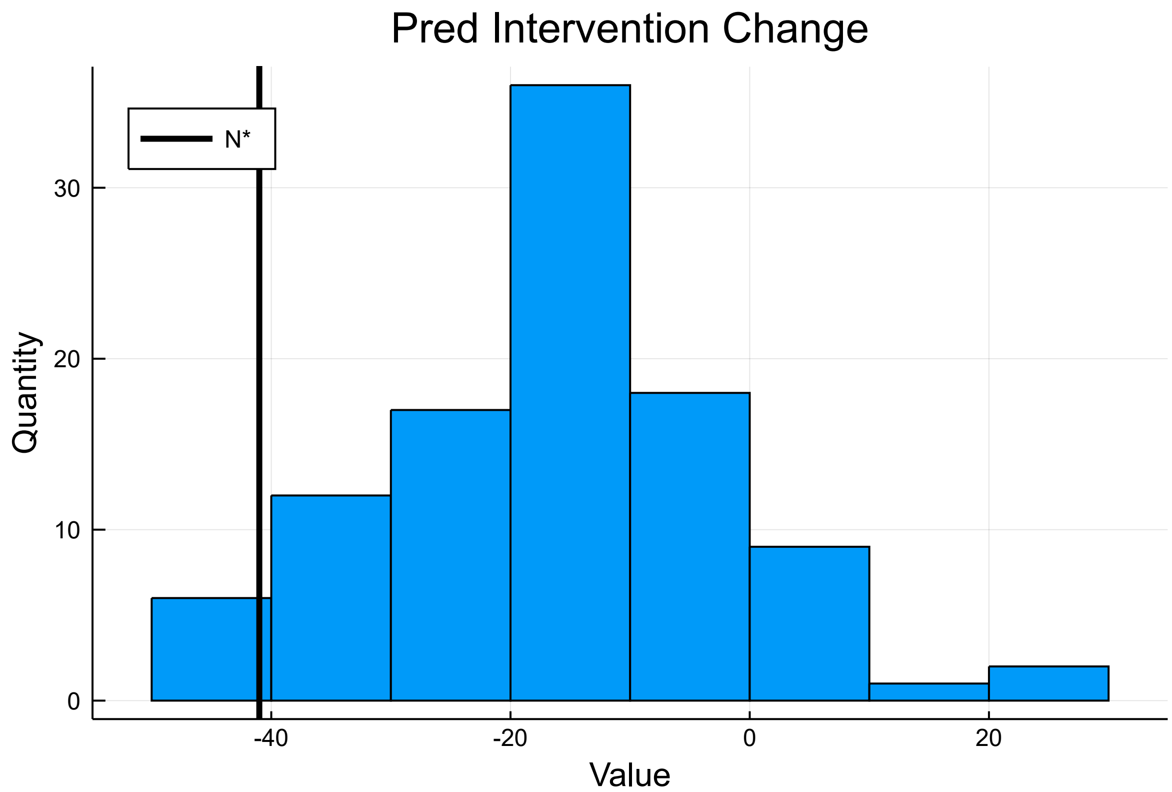
Out[81]:



In [83]:

```
histogram(pred_diff,
        title = "Pred Intervention Change",
        xlabel = "Value",
        ylabel = "Quantity",
        label = false,
        seriestype = :histogram,
        bins = 10)
plot!([-41], seriestype = :vline, color = "black", lw = 3, label = "N*", legend = :topleft)
```

Out[83]:



In [73]:
```
mean(prey_diff), mean(pred_diff)
```

Out[73]:
```
(97.33661198374551, -16.009912235861584)
```

Prey prediction distribution

In [77]:
```
fit_mle(Normal, convert(Array{Float64,1},prey_diff))
```

Out[77]:
```
Normal{Float64}(μ=97.33661198374551, σ=19.56318942561998)
```

Prey counterfactual prediction $N \sim (97.34, 19.6)$

Pred prediction distribution

In [78]:
```
fit_mle(Normal, convert(Array{Float64,1},pred_diff))
```

Out[78]:
```
Normal{Float64}(μ=-16.009912235861584, σ=14.353042894440632)
```

Predator counterfactual prediction $N \sim (-16, 14.4)$

From our infered traces, we can see that the mean difference for prey and pred after the interventions are 97 and -16 respectively. As the results generally follow a normal distribution, these values are our predictions in the abduction-action-prediction algorithm. Additionally we can actually add a variance to these predictions as well, to provide a sense of our uncertainty about the prediction.

These predictions are slightly different from the ground truth values of 137 and -41, however we can see that they are within two stardard deviations of the mean.

```
In [ ]:
```