Complementary notebook for ShoupModel.jl

This File

The purpose of this notebook is to accompany the ShoupModel.jl file, acting as supportive documentation for using the model. The model is based on the 2006 paper by Donal Shoup ***Cruising for Parking***

The Paper

In the paper, Shoup(2006) presents a model which seeks to capture how the price-ratio between curb-side/off-street parking, fuel cost, and an individual's value of time impacts the incentive for cruising. The paper concludes that when curb-side parking is underpriced, it creates an incentive for individuals to cruise for parking, resulting in congestion, air pollution. Bringing the price of curb-side parking in-line with the off-street parking price can consequently yield a triple-dividend: reducing search times, reduce congestion, and raise revenue to reduce the deadweight loss of other forms of taxation.

The Model

The core model, as presented in the paper, has seven variables:

- p - price of curb-side parking($/h)
- m - price of off-street parking ($/h)
- t - parking duration (h)
- c - time spent searching for parking at the curb (h)
- f - fuel cost of cruising ($/h)
- n - number of people in the car (persons)
- v - value of time spent cruising ($/h/person)

From these variables, a set of four relationships are established:

Equation	Definition
$t(m-p)$	`money saved by parking at the curb`
fc	`money cost of cruising for curb parking`
nvc	`monetized cost of time spent cruising for curb parking`
$fc + nvc = c(f+nv)$	`money and (monetized) time cost of cruising for curb parking`

First, the price money saved is the amount of time you wish to park multiplied by the hourly price difference between curb-side and off-street parking. This quantity represents the benefit, or the potential consumer surplus if the agent immediately found a space to park on the curb. On the other hand, the agent incurs two costs when searching for parking, the fuel cost, fc, and the monetized cost of time, nvc. The combined cost of these two would be the total cost incurred by an individual who is cruising.

The equilibrium cruising time is where the cost of cruising equates the potential benefits, that's to say:

$$c^*(f + nv) = t(m - p)$$

$$c^* = \frac{t(m-p)}{f+nv}$$

From the above, one can see that cities can impose several strategies to tackle the issue of cruising for parking.

1. If $m = p$, then there will no longer be an incentive to park on the curb. This can either be achieved by increasing the curb-side parking fee, or increase the amount of off-street parking such that m reduces to the same level as p.
2. Fuel taxes or emission permits could increase the cost of fuel and consequently increase the cost of cruising. In effect, this would reduce cruising however, unlikely to eliminate it, as it doesn't tackle the root cause of the issue..
3. Similarly to increases in increases in fuel cost, policy to promote carpooling, or secondary vehicle taxes may increase n and reduce cruising times.

Julia Implementation

Based on the simple model outlined above, one can outline a basic agent based model, where agents arrive to a curb-side parking location. If curbside parking is available, they will immediately park on the curb given that $p \leq m$. If no location is available, the agent will cruise for a maximum of time of c^*. If the agent has not been able to find a location to park within c^* minutes, it will park off-street. When an available parking spot on the curb opens up, all of the agents which are currently cruising for parking, are equally likely to occupy the available slot.

Variable list

To enable a degree of heterogeneity between agents, almost all model inputs can be provided as distributions or absolute values. Furthermore, to due to a lack of information on certain variables, such as how frequently people are looking for parking, a few additional variables have been added to the model.

Variable	Definition	Type

p	Price of curb-side parking ($/h)	Union{Real, Distribution}
m	Price of off-street parking ($/h)	Union{Real, Distribution}
t	Parking duration (h)	Union{Real, Distribution}
f	Fuel cost when cruising ($/h)	Union{Real, Distribution}
n	People in the car (persons)	Union{Real, Distribution}
v	Value of time ($/h/person)	Union{Real, Distribution}
ar	Arrival rate (arrivals/min)	Union{Real, Distribution}
cpk	Number of available curb-side spaces	Int64
mint	Minimum parking duration (h)	Real
minc	Minimum time spent coasting (h)	Real
minv	Minimum value of time ($/h/person)	Real
model_time	Total simulation period (minutes)	Int64
init_occup	Initial curb-side occupancy rate [0,1]	Float64

The arrival rate and other limiting variables need to be specified to ensure convergence. Since the model solves minute-by-minute, an arrival rate of more than 1 arrival per minute is not supported. However, if an arrival rate of more than one vehicle per minute is desired, this can be achieved by increasing the **model_time** and adjusting the other input parameters accordingly to reflect the alteration to the time horizons.

All input parameters into the model are also optional parameters, with each value being assigned a default value and classified in to one of 3 parameter groupings.

Variable	Default Value	Parameter Grouping
p	1.0	Characteristic
m	8.0	Characteristic
t	Distributions.Normal{Float64}(μ=1.0, σ=0.5)	Preference
f	1.0	Characteristic
n	Distributions.Binomial{Float64}(n=3, p=0.8)	Preference
v	Distributions.Normal{Float64}(μ=40.0, σ=5.0)	Preference
ar	Distributions.Bernoulli{Float64}(p=0.2)	Characteristic
cpk	Number of available curb-side spaces	Characteristic
mint	0.1	Characteristic
minc	0.0	Characteristic
minv	0.0	Characteristic
model_time	720	Model
init_occup	0.0	Model

NOTE: The default values represent arbitrary but plausible values.

The model groupings serve no computational purpose, but were included as an aid to organise the parameter inputs in-case of an expansion of the model in future.

Model setup

The parameters are specified by the `init_params()` function, where all input parameters are entered as optional variables. The function returns a a tuple conatining each of the parameter groupings.

```
(CharParams, PrefParams, ModelParams)
```

Each parameter group is another nested tuple containing the relevant variables (see table above). The parameters are then passed to `init_dataframe()` which initialises a dataframe containing the characteristics of each agent, including the arrival time. In cases where a distribution is passed, the characteristics of the agent are independently sampled from the distrubtion specified. Sample dataframe output below.

5 rows × 9 columns

	t	n	v	p	m	f	ar	cpk	mint
	Float64	Float64	Float64	Float64	Float64	Float64	Float64	Float64	Float64
1	0.482728	3.0	0.482728	1.0	8.0	1.0	0.0	8.0	0.1
2	1.47324	2.0	1.47324	1.0	8.0	1.0	0.0	8.0	0.1
3	1.76026	3.0	1.76026	1.0	8.0	1.0	0.0	8.0	0.1
4	0.879491	3.0	0.879491	1.0	8.0	1.0	1.0	8.0	0.1
5	1.26833	2.0	1.26833	1.0	8.0	1.0	0.0	8.0	0.1

5 rows × 8 columns

	minc	minv	c	psav	ccost	mct	arrt	tmin
	Float64	Float64	Float64	Float64	Float64	Float64	Int64	Float64
1	0.0	0.0	2.04405	14.3084	14.5785	59.0	1	122.643
2	0.0	0.0	1.33751	9.36254	3.12643	180.0	8	80.2504
3	0.0	0.0	0.300854	2.10598	0.572393	221.0	9	18.0512
4	0.0	0.0	1.16924	8.18465	3.90346	126.0	10	70.1541
5	0.0	0.0	0.524071	3.6685	1.34802	163.0	17	31.4443

Based on the parameters set, the potential savings from parking on the curb, and the hourly cruising costs are calculated. These values are stored in *psav* and *ccost* respectively and are calculated based on *c* (c^* in the paper). The maximum cruising time and desired cruising duration, in minutes, are given by *mct* and *tmin* respectively. Lastly, *arrt* is the iteration which the agent starts looking for parking which is determined based on the arrival rate parameter, *ar*.

The dataframe and input parameters are then passed to `run_simulation()`, which runs the model over the specified time horizon. The model returns an array with *model_time* rows, populated with a struct of type `ParkState`, where each parkstate contains 8 variables. The struct includes:

Variable	Definition
curb_par_current	Current number of agents parked on the curb
offs_park_current	Current number of agents parked off-street
cruising_curent	Current number of vehicles cruising
curb_park_total	Total number of vehicles parked on the curb
offs_park_total	Total number of vehicles parked off-street
cruising_total_time	Total time spent cruising (hours)
curb_revenue	Revenue for curb-parking provider ($)
offs_revenue	Revenue for off-street parking provider ($)

By using the `as_matrix()` function, the simulation output can be converted into a matrix of dimension [*model_time*, 8]. When in matrix format, the each column will correspond to one of the variables above, where the order of the columns will correspond with the order of the elements in the struct.

Monte-carlo

When model inputs are passed as distributions, it is more insightful to run a monte-carlo to get a sence of the distribution of the outcome variables of interest. To support this, `mc_simulation()` runs the simulation n_{mc} times. For the monte-carlo simulation, the default output is a 3D array of size [*model_time*, 8, n_{mc}]. Since the dataframe is generated for each model run, the function does not require a dataframe input, only the parameters and the number iterations.

Example

In this example, we'll look at how different pricing policies may impact congestion, and air pollution. For the sake of comparability, we'll be using the figures for the 2020 Honda Civic as a representative city car.

Running a single simulation

First step is to set the characteristics of the desired vehicle, a 2020 Honda Civic in this case.

```
begin
    #Setting vehicle emissions, coasting speed, and fuel efficiency (L/hour
coasting)
    co2_kgkm = 0.11
    nox_kgkm = 1.49e-4
    coasting_speed_kmh = 8
    fuel_lh = 0.093*coasting_speed_kmh
    nothing
end
```

Subsequently, the rest of the model parameters can be set. In this case, all the model parameters are called explicitly, however, a parameter does not need to be explicitly set if the default value is desired.

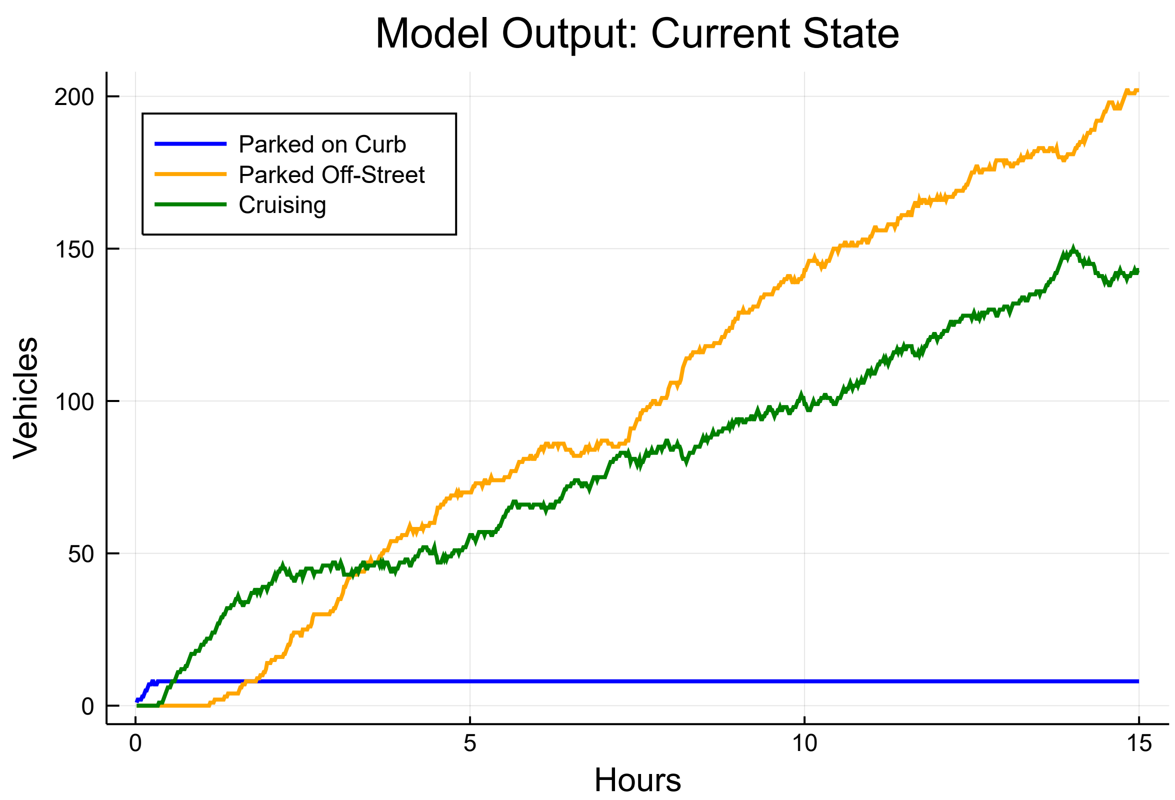
```
#Setting model parameters
pparams, cparams, mparams = init_params(p          = 1.0,
                                        m          = 13.25,
                                        t          = Normal(1.5,5),
                                        f          = fuel_lh,
```

```
                                    n          = Binomial(2,0.5),
                                    v          = Normal(40,5),
                                    ar         = Bernoulli(0.5),
                                    cpk        = 8,
                                    mint       = 0.1,
                                    minc       = 0.0,
                                    minv       = 10,
                                    model_time = 900,
                                    init_occup = 0.0);

```

Once the model parameters have been stored in our parameter variables, we can generate the dataframe and run the simulation.

```
begin
    #Setting model parameters
    model_df = init_dataframe(pparams, cparams, mparams)

    #Running simulation
    model_results = run_simulation(model_df, pparams, cparams, mparams)
    nothing
end
```

We can not plot the results with respect to time.



The above graphs shows the state which agents are in along the time-horizon which the model solves for. The green line shows the number of vehicles which are currently cruising to look for parking. Similarly, the blue and orange lines represent the number of vehicles which are currently parked either on the curb or off-street respectively.

Running a many simulations

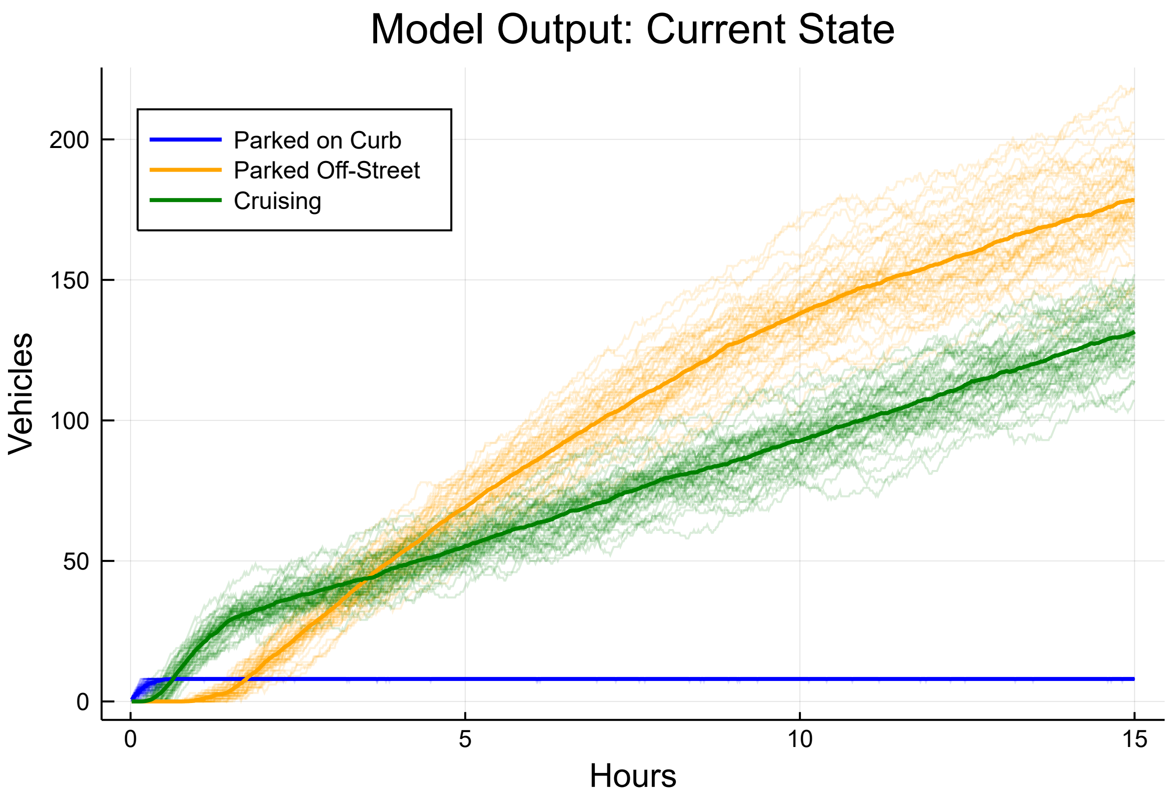
Arguably, running a single simulation is not particularlyn useful, as we are pulling from several distributions, it could be that we are just getting a tail-case event. Running a monte-carlo, we can get a better sense of what range of results one may expect from the model output.

We are going to use the same parameter settings as above, however, we will now also specify the number of times we want to run the simulation.

model iterations: 50

```
model_results_mc = mc_simulation(pparams, cparams, mparams, n=model_iterations);
```

Plotting the results, we get the following:



curb-side transparency: 0.15

off-street transparency: 0.15

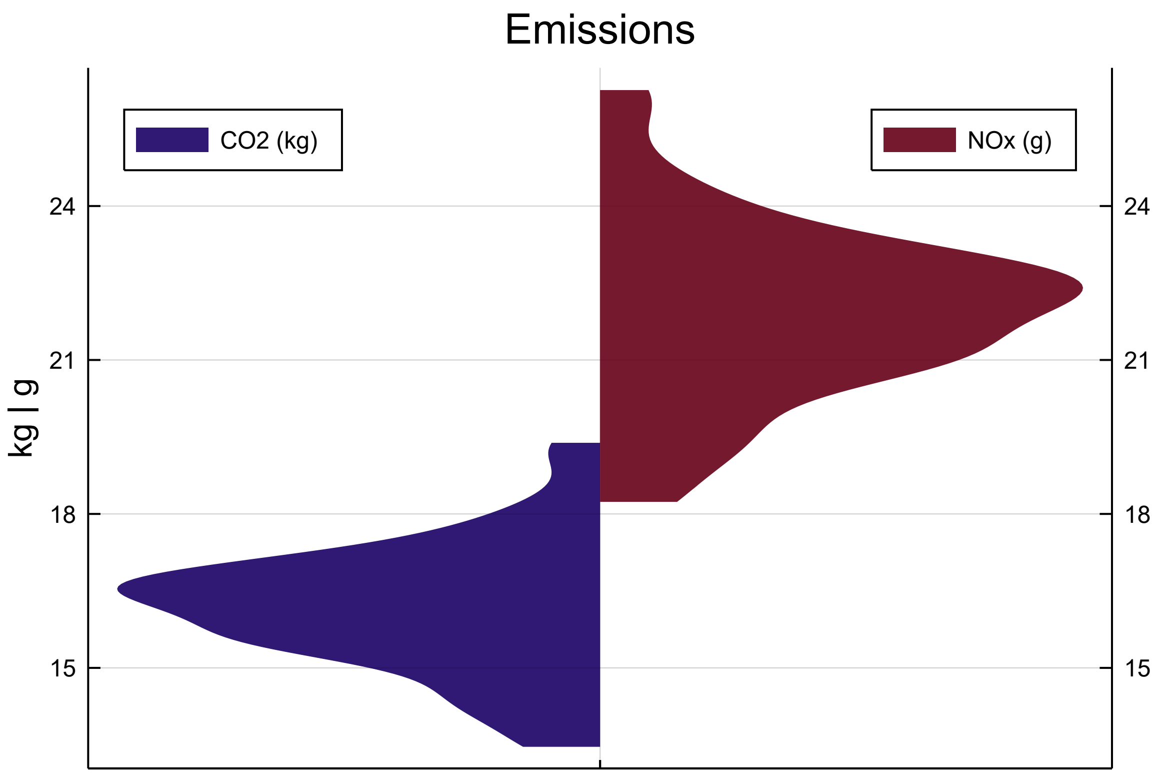
cruising transparency: 0.15

Emissions

Using the vehicle emission data specified earlier (0.11 kg CO_2/km and 0.000149 kg NO_x/km) and assuming an average cruising speed of 8 km/h, we can easily derive the emissions generated from coasting.

```
begin
    #Creating the dataframe
    emissions = calculate_emissions(model_results_mc;
                                    emission_co2=co2_kgkm,
                                    emission_nox=nox_kgkm) |>
                DataFrame

    #Multiplying nox by 1,000 to be easier to compare to CO2 figures
    emissions[:nox]*=1000    #g/km
```

```
    emissions[:label] = ""
    nothing
end
```



The variance in the emission distribution is determined by variances in cruising time. On the left the CO_2 emissions in kg, and on the right are the NO_x emissions in g.

Cruising time

- Total distance cruised
- Total time spent cruising
- What does this equate to in terms of distance frmo LA to NY
- Disclaimer that an infinite street is assumed.

Conclusion

- Summary of the paper
- Overview of the results from the model
- Potential improvements
- Limitaitons

Appendix

Limitations and improvements

1. Hourly prices seldom work additively over longer parking stays.

2. Agents are currently miopic and do not have a prior expectation of how long it will take to find parking.
3. Street is assumed to be limitless and can host an infinite amount of cruisers
4. Assumes static variables/distributions throughout the modelling period. i.e. doesn't account for rush-hour etc.

Function for calculating emissions

```
function calculate_emissions(results; emission_co2=0.11, emission_nox=1.49e-4,
coast_speed_kmh=8)
    #Calculate the total amount of time spend coasting
    hours_coasting = results[end,6,:]/60

    #Calculate emissions
    emissions_co2 = (hours_coasting*coast_speed_kmh)*emission_co2 #Kg of CO2
    emissions_nox = (hours_coasting*coast_speed_kmh)*emission_nox #Kg of NOx

    #Return named tuple with results
    return (co2=emissions_co2, nox=emissions_nox)
end;
```