```
from scipy.stats import multivariate_normal

# number of draws for the sampler
n_runs = 4_000

verbose = False
start = time.time()

# counters
counter_solved = 0 # model was sovable
counter_kalman = 0 # kalman filter did not fail
counter_accp = 0 # draw was accepted into posterior

# reset params in the model
mod.free_param_dict.update(mod_params)

# get params, variables and shocks as lists
shock_names = [x.base_name for x in mod.shocks]
state_variables = [x.base_name for x in mod.variables]
model_params = list(mod.free_param_dict.keys())

# set kalman filter observed variables
observed_vars = ['y', 'n', 'r', 's']
_ = [item for item in observed_vars if item not in state_variables]
assert len(_) == 0, f"{_} not in state variables"

# get posterior output list
param_posterio_list = {item: [mod.free_param_dict[item]] for item in model_par
shock_posterior_list = {item: [.1] for item in shock_names}
loglike_list = [-100]

# get simga and scaling factor for the random walk law of motion
p, s = get_Sigma(prior_dist, mod_params, shock_names)
  _params   = np.array(list(p.values())) #
  _shocks   = np.array(list(s.values()))

  _params   = np.zeros(len(p)) + 1
  _shocks   = np.zeros(len(s)) + 1


scaling_factor = .4

# get final ouput
output_dict = {}

for i in range(0, n_runs):
```

```
printProgBar(i, n_runs-1, prefix='Progress')

# set dict to capture results of this run
draw_dict = {
    'log_like_list': None,
    'log_like_sum': None,
    'is_solved': False,
    'ratio': None,
    'omega': None,
    'is_KF_solved': False,
    'KF_execption': None,
    'is_accepted': False,
    'parameters': {
        'prior': {item: None for item in model_params if item in prior_di
        'prior_pdf_p': {item: None for item in model_params if item in pr
        'posterior': {item: None for item in model_params if item in prio
    },
    'shocks': {
        'prior': {item: None for item in shock_names},
        'prior_pdf_s': {item: None for item in shock_names if item in pri
        'posterior': {item: None for item in shock_names}
    }
}
# current posterior
old_posterior_p = {item: vals[-1] for item, vals in param_posterio_list.it
old_posterior_s = {item: vals[-1] for item, vals in shock_posterior_list.
old_loglike = loglike_list[-1]

# save posterior information to output dict
draw_dict['parameters']['posterior'] = old_posterior_p
draw_dict['shocks']['posterior'] = old_posterior_s


# sample from priors according to random walk law of motion
# a multivariate normal distribution wit  standard deviation

# prior for parameters
prior = np.array(list(old_posterior_p.values()) + multivariate_normal(list

# prior for shocks
shocks = np.array(list(old_posterior_s.values()) + multivariate_normal(lis

# put priors into dictionary for further process
prior, shocks = dict(zip(old_posterior_p.keys(), prior)), dict(zip(old_po

draw_dict['parameters']['prior'].update(prior)
```

```
draw_dict['shocks']['prior'].update(shocks)

# update model with new parameters and shocks
mod.free_param_dict.update(prior)
mod.shock_priors.update(shocks)

# solve model for new steady state and transition matrix T
# if model is not solved discard and proceed to nex iteration
is_solved, mod = solve_updated_mod(mod, verbose=verbose, model_is_linear=
if not is_solved:
    output_dict[i] = draw_dict
    continue
else:
    draw_dict['is_solved'] = True
    counter_solved += 1

# get Kalman filter matrices
T, R = mod.T.values, mod.R.values
H, Z, T, R, QN, zs = set_up_kalman_filter(R=R, T=T, observed_data=train[o
                                           shock_names=shock_names, shocks

# set up Kalman filter
kfilter = KalmanFilter(len(state_variables), len(observed_vars))
kfilter.F = T
kfilter.Q = QN
kfilter.H = Z
kfilter.R = H

# run Kalman filter
try:
    saver = Saver(kfilter)
    mu, cov, _, _ = kfilter.batch_filter(zs, saver=saver)
    ll = saver.log_likelihood

    # catch -math.inf values in log_likelihood, meaning that the filter
    if len([val for val in ll if val == -math.inf]) >0:
        output_dict[i] = draw_dict
        continue

    # sum-up individual log-likelihoods in order to obtain the ll for thi
    new_loglike = np.sum(ll)
    loglike_list.append(new_loglike)

    # save information in dict and update counters
    draw_dict['log_like_sum'] = new_loglike
    draw_dict['log_like_list'] = ll
```

```
        draw_dict['is_KF_solved'] = True
        counter_kalman += 1

    # catch possible errors in KFilter for debugging
    except Exception as e:
        draw_dict['KF_execption'] = e
        output_dict[i] = draw_dict
        continue

    # Metropolis Hastings MCMC sampler

    # get prior likelihood
    prior_pdf_p = get_arr_pdf_from_dist(prior, prior_dist)
    prior_pdf_s = get_arr_pdf_from_dist(shocks, prior_dist)
    prior_pdf = np.append(prior_pdf_p, prior_pdf_s)

    # save prior likelihood
    draw_dict['parameters']['prior_pdf_p'] = dict(zip(prior.keys(), prior_pdf
    draw_dict['shocks']['prior_pdf_s'] = dict(zip(prior.keys(), prior_pdf_s))

    # get current posterior likelihood
    mask_old_post = np.append(get_arr_pdf_from_dist(old_posterior_p, prior_di

    # compare current prior likelihood with the likelihood of the most recent
    ratio = (new_loglike * prior_pdf / old_loglike * mask_old_post).mean()
      = min([ratio, 1])
    # save ratios
    draw_dict['ratio'] = ratio
    draw_dict['omega'] =

    # MH sampler random acceptance, based on uniform distribution U(0,1)
    random = np.random.uniform(0, 1)
    if random <=   :
        counter_accp += 1.
        draw_dict['is_accepted'] = True

        # update param posterior
        for key in prior.keys():
            param_posterio_list[key].append(prior[key])

        # update shock posterior
        for key in shock_posterior_list:
            shock_posterior_list[key].append(shocks[key])

    else:
        # leave posterior unaltered and restart
```

```python
            is_accepted = False

        # save output
        output_dict[i] = draw_dict


# print stats
print('\nloop ran for', (time.time() - start) / 60, 'minutes')
print('\nsolver rate', counter_solved/n_runs)
print('\nacceptance rate', counter_accp/counter_solved)
```