basic_example

January 3, 2020

```
In [30]: using PyPlot
         using JuMP, Ipopt
```

```
In [38]: include("../src/model.jl");
         include("../src/diagnostics.jl");
         include("../src/optimization.jl");
         include("../src/plotting.jl");
```

1 Optimizating the Global Climate Action Policy Portfolio

Physical Climate Model Parameters

```
In [39]: # Model domain
         dt = 1. # 1-year timestep, can make longer to speed up the model
         t = Array(2020:dt:2200);

         ECS = 3.0; # "Best-guess equilibrium climate sensitivity"
```

Economic parameters Assume damages of 2% of global world product (GWP) for 3°C of warming (similar to DICE damage function).

```
In [40]: GWP = 100. # global world product (trillion $ / year)

          = 0.02*GWP/(3.0)^2 # damages (trillion USD / year / celsius^2)
         utility_discount_rate = 0.025 #  (relative low value from Stern review)

         # Control technology cost scales, as fraction of GWP (cost scale is for full deployment
         reduce_cost = 0.01*GWP;
         remove_cost = 0.02*GWP;
         geoeng_cost = 0.05*GWP;
         adapt_cost = 0.03*GWP;
```

No-policy baseline scenario We begin with a reference no-policy baseline scenario in which CO2 concentrations increase at a rate determined by the emissions rate $q(t)$, which here a constant $q_0 = 5$ ppm/year (second argument of `baseline_emissions()` function) until 2060 (third argument), at which point emissions are assumed to decrease to zero over the next 40 years (fourth argument).

All of these parameters are easily modified in the `baseline_emissions()` function below.

```
In [41]: baseline_emissions_scenario = baseline_emissions(t, 5., 2080., 40.)

         economics = Economics(
             , utility_discount_rate,
             reduce_cost, remove_cost, geoeng_cost, adapt_cost,
             0., 0., 0., 0., # Assumed initial condition of zero control deployments in 2020
             baseline_emissions_scenario
         );
```

1.0.1 Initialize model

```
In [42]: ensemble = Dict{String, ClimateModel}();

         name = "Example configuration";
         present_year = 2020.

         # Arbitrary initial guess of climate control policies, to be optimized later!
         controls = init_zero_controls(t);

         # Create instance of idealized integrated assessment climate model
         model = ClimateModel(name, ECS, t, dt, controls, economics, present_year);

         # Calculate and print the social cost of carbon (defined by the
         # climate damages due to an extra metric ton of CO2 emitted in 2020)
         print(string("Social cost of carbon: \$", SCC(model)))
```

```
Social cost of carbon: $25.13
```

1.1 Model optimization

We now calculate the optimal trajectories of climate control deployments α by minimizing an objective function (also known as a "cost function") under a set of physical and policy constraints. The physical constraints are that the fractional deployments of climate controls are between 0 (no deployment) and 1 (full deployment). The policy constraints are that the climate controls begin at 0 fraction deployment in 2020 and that deployments can only change at a maximum rate of $\partial\alpha \,/\, \partial t < 1/30$ (i.e. from zero to full deployment in 30 years).

The model currently supports three optimizations options, representing different assumed cost functions or additional policy constraints: - `net_cost`: in this scenario, the model optimizes the total net discounted cost of handling climate change, including both the direct costs $\beta\,\delta T_{\alpha}^{2}\,(1-\chi)$ of (controlled) damages from climate impacts and the costs $\sum C_{\alpha} f(\alpha)$ of deploying climate controls.

- `temp`: in this scenario, the model finds the lowest total discounted cost of climate controls which results in temperatures that remain below a specified temperature goal. The temperature goal is given as an additional parameter in degrees Celsius, e.g. `optimize_controls!(model, maxslope = maxslope, obj_option = "temp", temp_goal = 1.5)`.

- `budget`: in this scenario, the model finds the combination of discounted climate control investments which, given a specified budget, provides the lowest (controlled) damages from cimate impacts. The budget is given as an additional parameter in trillions of USD, e.g. `optimize_controls!(model, maxslope = maxslope, obj_option = "budget", budget = 10.)`.

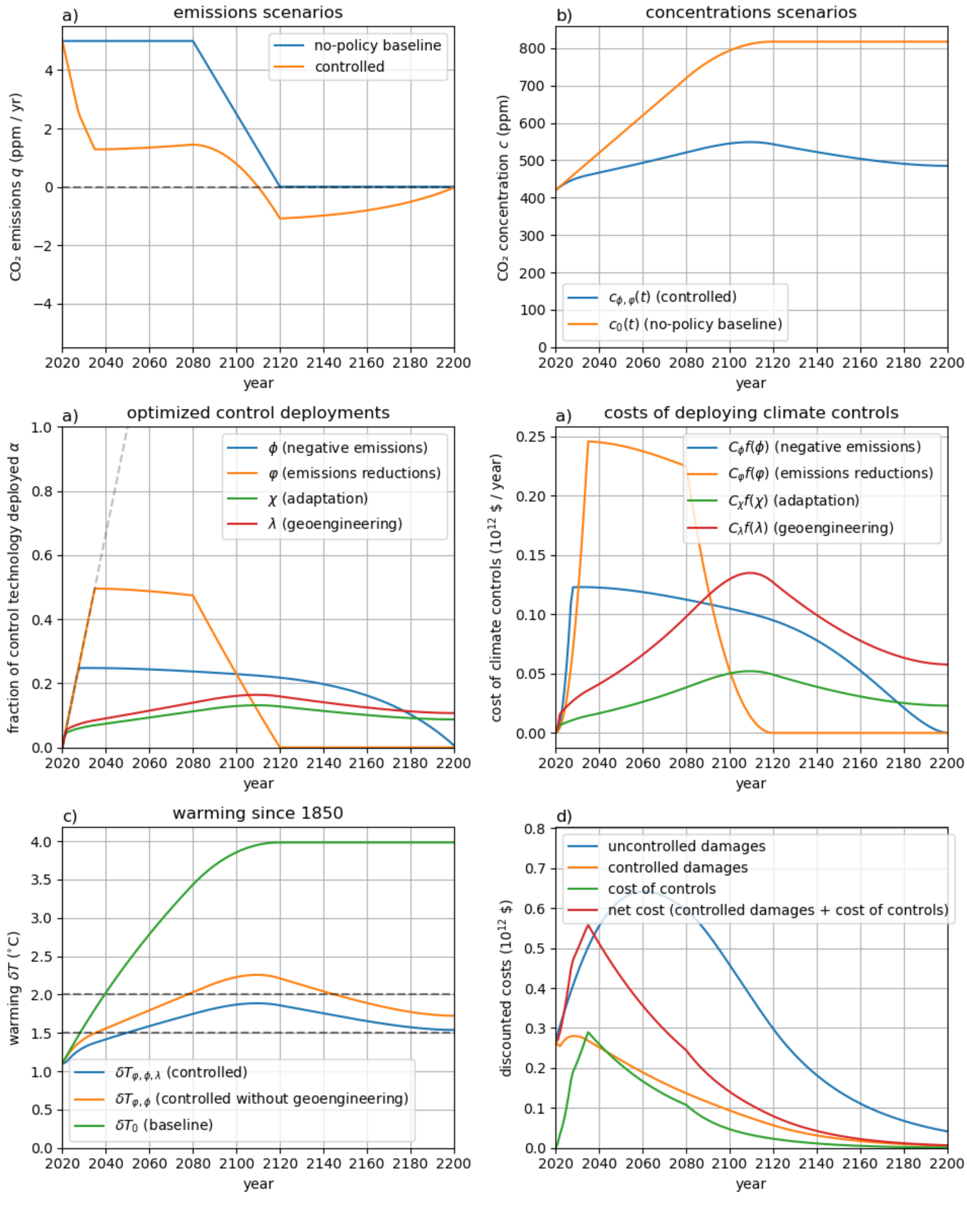
In [43]:
```
maxslope = 1. /30.
optimize_controls!(model, maxslope = maxslope, obj_option = "net_cost");
```

```
This is Ipopt version 3.12.10, running with linear solver mumps.
NOTE: Other linear solvers might be more efficient (see Ipopt documentation).

Number of nonzeros in equality constraint Jacobian...:     2796
Number of nonzeros in inequality constraint Jacobian.:        0
Number of nonzeros in Lagrangian Hessian.............:     1981

Total number of variables............................:     1621
                     variables with only lower bounds:        0
                variables with lower and upper bounds:     1440
                     variables with only upper bounds:        0
Total number of equality constraints.................:      901
Total number of inequality constraints...............:        0
        inequality constraints with only lower bounds:        0
   inequality constraints with lower and upper bounds:        0
        inequality constraints with only upper bounds:        0

iter    objective    inf_pr   inf_du lg(mu)  ||d||  lg(rg) alpha_du alpha_pr  ls
   0  1.0625947e+01 5.00e+00 3.78e-01  -1.0 0.00e+00    -  0.00e+00 0.00e+00   0
   1  2.0998988e+01 3.08e+00 4.96e-01  -1.0 3.02e+02    -  4.82e-01 3.85e-01f  1
   2  2.1453796e+01 2.25e+00 3.40e+00  -1.0 6.53e+01    -  4.62e-01 2.70e-01f  1
   3  3.5660198e+01 2.58e-14 3.14e-02  -1.0 5.66e+01    -  1.00e+00 1.00e+00f  1
   4  3.3716163e+01 2.58e-14 1.87e+00  -2.5 1.12e+01    -  8.36e-01 1.00e+00f  1
   5  3.2586844e+01 2.71e-14 2.47e-03  -2.5 1.74e+01    -  1.00e+00 1.00e+00f  1
   6  3.2092212e+01 2.49e-14 4.30e-02  -3.8 1.62e+01    -  8.34e-01 1.00e+00f  1
   7  3.2016767e+01 2.31e-14 1.12e-04  -3.8 6.36e+00    -  1.00e+00 1.00e+00f  1
   8  3.1994974e+01 2.62e-14 6.85e-03  -5.7 2.33e+00    -  8.15e-01 1.00e+00f  1
   9  3.1991243e+01 2.18e-14 5.53e-04  -5.7 9.74e-01    -  9.66e-01 1.00e+00f  1
iter    objective    inf_pr   inf_du lg(mu)  ||d||  lg(rg) alpha_du alpha_pr  ls
  10  3.1990262e+01 2.07e-14 4.86e-08  -5.7 4.33e-01    -  1.00e+00 1.00e+00f  1
  11  3.1989943e+01 2.66e-14 3.84e-05  -8.6 1.67e-01    -  9.68e-01 1.00e+00f  1
  12  3.1989876e+01 2.13e-14 3.51e-11  -8.6 3.54e-02    -  1.00e+00 1.00e+00f  1
  13  3.1989859e+01 1.95e-14 1.34e-12  -8.6 7.27e-03    -  1.00e+00 1.00e+00f  1
  14  3.1989855e+01 2.75e-14 1.86e-14  -8.6 1.13e-03    -  1.00e+00 1.00e+00f  1
  15  3.1989854e+01 2.26e-14 6.91e-16  -8.6 5.43e-04    -  1.00e+00 1.00e+00h  1
  16  3.1989853e+01 2.66e-14 3.39e-15  -9.0 2.86e-04    -  1.00e+00 1.00e+00h  1

Number of Iterations....: 16
```

```
                                   (scaled)                 (unscaled)
Objective...:   3.1989853369895787e+01    3.1989853369895787e+01
Dual infeasibility...:   3.3902964222457597e-15    3.3902964222457597e-15
Constraint violation...:   2.6645352591003757e-14    2.6645352591003757e-14
Complementarity...:   3.4245891262791444e-09    3.4245891262791444e-09
Overall NLP error...:   3.4245891262791444e-09    3.4245891262791444e-09


Number of objective function evaluations             = 17
Number of objective gradient evaluations             = 17
Number of equality constraint evaluations            = 17
Number of inequality constraint evaluations          = 0
Number of equality constraint Jacobian evaluations   = 1
Number of inequality constraint Jacobian evaluations = 0
Number of Lagrangian Hessian evaluations             = 16
Total CPU secs in IPOPT (w/o function evaluations)   =      0.232
Total CPU secs in NLP function evaluations           =      0.305

EXIT: Optimal Solution Found.
  0.554343 seconds (1.45 M allocations: 72.572 MiB, 6.15% gc time)
```

1.1.1 Plotting the optimal solution state

In [44]:
```
plot_state(model)

# Individual panels of the figure can be modified using PyPlot commands
# (see https://github.com/JuliaPy/PyPlot.jl)
subplot(3,2,3)
plot([2020., 2020. + 1. /maxslope], [0.,1.], "k--", alpha=0.25) # add a slope showing t
```




```
Out[44]: 1-element Array{PyCall.PyObject,1}:
 PyObject <matplotlib.lines.Line2D object at 0x143956a20>
```