# L1 with Shape Constraints

In[1506]:=
```
x = 0; Remove["Global`*"]; DateList[Date[]] // Most
```

Out[1506]=
```
{2020, 4, 29, 20, 47}
```

# Log utility and C-D production

We analyze the discrete-time growth model with Cobb-Douglas production function and quadratic utility.

We choose parameters so that the steady state is k=1.

In[1507]:=
```
β = 95/100; α = 1/4; ftrue[x_] = x + A x^α;
A=(1-β)/(α β); kss=1; css = ftrue[1]-1;
```

We choose the log utility function

In[1509]:=
```
utrue[x_] = Log[x];
```

## Choose production and utility function

In[1510]:=
```
f[x_] = ftrue[x];  u[c_] = utrue[c];
```

Define Euler equation error function

In[1511]:=
```
Eulerf[x_] = u'[cf[x]] - β u'[cf[f[x] - cf[x]]] × f'[f[x] - cf[x]]
```

Out[1511]=

$$\frac{1}{\text{cf}[x]}-\frac{19\left(1+\frac{1}{19\left(\frac{4x^{1/4}}{19}+x-\text{cf}[x]\right)^{3/4}}\right)}{20\,\text{cf}\left[\frac{4x^{1/4}}{19}+x-\text{cf}[x]\right]}$$

# Choose domain, approximation, and nodes

Set the range over which we solve the problem

```
In[1512]:= xmin = .2;  xmax = 2.0;
```

Choose approximation

```
In[1513]:= degreecf = 9;
cf[x_] = Sum[a[i] x^i, {i, 0, degreecf}];
```

Choose approximation nodes

```
In[1515]:= numpts = 17;
delx = (xmax - xmin) / (numpts - 1);
nodes = Table[x, {x, xmin, xmax, delx}]

Out[1517]= {0.2, 0.3125, 0.425, 0.5375, 0.65, 0.7625, 0.875, 0.9875,
 1.1, 1.2125, 1.325, 1.4375, 1.55, 1.6625, 1.775, 1.8875, 2.}
```

Display the Euler equation error function (don’t look too closely; it is ugly)

In[1518]:=
```
Eulerf[x];
```

# Create the optimization problem

## Create simple initial guess for a

The following creates an initial guess for the a[i] coefficients that is zero at k=0, and is css at k=1

In[1519]:=
```
varsa = Variables[cf[7^(1/7)]]
```

Out[1519]=
```
{a[0], a[1], a[2], a[3], a[4], a[5], a[6], a[7], a[8], a[9]}
```

In[1520]:=
```
inita = Table[0, {Length[varsa]}];
inita[[2]] = css // N;
vvarsa = {varsa, inita} // Transpose
```

Out[1522]=
```
{{a[0], 0}, {a[1], 0.210526}, {a[2], 0}, {a[3], 0},
 {a[4], 0}, {a[5], 0}, {a[6], 0}, {a[7], 0}, {a[8], 0}, {a[9], 0}}
```

vvarsa is the list of variables and initial conditions we feed to the optimizer.

## Replace equations with upper and lower bounds

If we pursued a nonlinear equation approach, we would try to solve the equation

Eulerf[k] = 0

for all k in the set of approximation nodes.

We do not do that because we may have more nodes than a[i] coefficients. Also, we want to add conditions to steer the solution in the right direction.

We replace the Euler equations with lower and upper bounds

In[1523]:=
```
EulerBnds = Table[-λl[i] ≤ Eulerf[nodes[[i]]] ≤ λu[i], {i, 1, Length[nodes]}];
```

We constrain the $\lambda$'s to be nonnegative

In[1524]:=
```
λbnds = Table[{λl[i] ≥ 0, λu[i] ≥ 0}, {i, 1, Length[nodes]}];
```

We want to create an optimization problem that pushes all the bounds as close to zero as possible.

## Define the objective in terms of the bounds

Our objective is the sum of the magnitude of the bounds (now you see why the $\lambda$'s must be nonnegative).

In[1525]:=
```
fitobj = Sum[λl[i] + λu[i], {i, 1, Length[nodes]}];
```

Our initial guesses for the $\lambda$'s are numbers that are large enough so that all the Euler equation error bounds are true at the initial guess.

In[1526]:=
```
varsλ = Variables[fitobj];
initλ = Table[100 000, {Length[varsλ]}];
vvarsλ = {varsλ, initλ} // Transpose
```

Out[1528]=
```
{{λl[1], 100 000}, {λl[2], 100 000}, {λl[3], 100 000},
 {λl[4], 100 000}, {λl[5], 100 000}, {λl[6], 100 000}, {λl[7], 100 000},
 {λl[8], 100 000}, {λl[9], 100 000}, {λl[10], 100 000}, {λl[11], 100 000},
 {λl[12], 100 000}, {λl[13], 100 000}, {λl[14], 100 000},
 {λl[15], 100 000}, {λl[16], 100 000}, {λl[17], 100 000}, {λu[1], 100 000},
 {λu[2], 100 000}, {λu[3], 100 000}, {λu[4], 100 000}, {λu[5], 100 000},
 {λu[6], 100 000}, {λu[7], 100 000}, {λu[8], 100 000}, {λu[9], 100 000},
 {λu[10], 100 000}, {λu[11], 100 000}, {λu[12], 100 000}, {λu[13], 100 000},
 {λu[14], 100 000}, {λu[15], 100 000}, {λu[16], 100 000}, {λu[17], 100 000}}
```

## Add shape constraints

We impose the requirement that there is positive savings at xmin and negative savings at xmax.

```
In[1529]:= consbnd = {f[xmax] ≥ cf[xmax] ≥ f[xmax] - xmax, 0 ≤ cf[xmin] ≤ f[xmin] - xmin};
```

Impose monotonicity at the approximation nodes

```
In[1530]:= ConsMono = Table[ cf'[nodes[[i]]] ≥ 0, {i, 1, Length[nodes]}];
```

## Collect all variables and constraints

Collect all variables

```
In[1531]:= vars = Union[vvarsa, vvarsλ];
```

Collect all constraints

```
In[1532]:= Constraints = Union[λbnds, ConsMono, EulerBnds, consbnd] // Flatten;
```

# Solve

Solve the optimization problem

In[1533]:=
```
{errsum, sola} = FindMinimum[{fitobj, Constraints}, vars]
```

Out[1533]=
```
{0.000340554, {a[0] → 0.0428438, a[1] → 0.348768, a[2] → -0.605764,
  a[3] → 1.15677, a[4] → -1.55843, a[5] → 1.41578, a[6] → -0.844387,
  a[7] → 0.315808, a[8] → -0.0670288, a[9] → 0.00615191, λl[1] → 1.76841 × 10^-7,
  λl[2] → 1.33367 × 10^-7, λl[3] → 1.40437 × 10^-6, λl[4] → 8.09838 × 10^-8,
  λl[5] → 1.33705 × 10^-7, λl[6] → 0.0000513749, λl[7] → 3.78552 × 10^-7,
  λl[8] → 8.10743 × 10^-8, λl[9] → 8.11184 × 10^-8, λl[10] → 3.53446 × 10^-6,
  λl[11] → 0.0000217609, λl[12] → 1.18514 × 10^-7, λl[13] → 8.12917 × 10^-8,
  λl[14] → 1.98148 × 10^-7, λl[15] → 0.0000362548, λl[16] → 9.65944 × 10^-8,
  λl[17] → 2.03279 × 10^-7, λu[1] → 1.67058 × 10^-7, λu[2] → 2.38453 × 10^-7,
  λu[3] → 8.71537 × 10^-8, λu[4] → 0.000110718, λu[5] → 2.37445 × 10^-7,
  λu[6] → 8.10702 × 10^-8, λu[7] → 1.08675 × 10^-7, λu[8] → 0.0000501108,
  λu[9] → 0.0000395757, λu[10] → 8.33016 × 10^-8, λu[11] → 8.12903 × 10^-8,
  λu[12] → 3.00678 × 10^-7, λu[13] → 0.000021683, λu[14] → 1.51401 × 10^-7,
  λu[15] → 8.11376 × 10^-8, λu[16] → 6.07801 × 10^-7, λu[17] → 1.48446 × 10^-7}}
```

# Verify solution quality

Check the constraints

```
In[1534]:= consbnd /. sola

Out[1534]= {True, True}

In[1535]:= EulerBnds /. sola

Out[1535]= {True, True, True, True, True, True, True,
  True, True, True, True, True, True, True, True, True, True}

In[1536]:= ConsMono /. sola

Out[1536]= {True, True, True, True, True, True, True,
  True, True, True, True, True, True, True, True, True, True}
```

Define consumption function implied by the solution

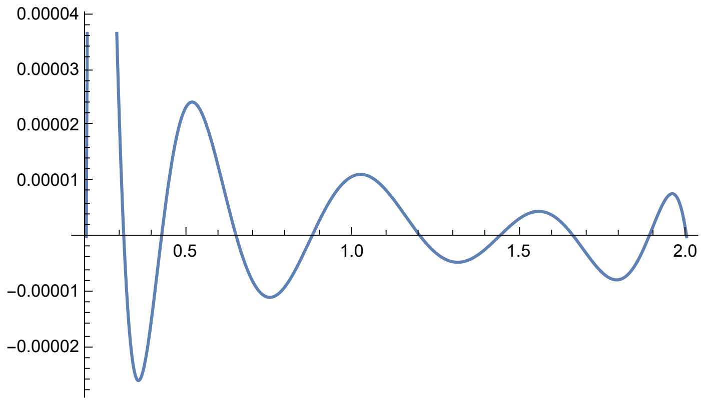
In[1537]:=
```
csol[x_] = cf[x] /. sola
```

Out[1537]=
$$0.0428438 + 0.348768\, x - 0.605764\, x^2 + 1.15677\, x^3 - 1.55843\, x^4 + 1.41578\, x^5 - 0.844387\, x^6 + 0.315808\, x^7 - 0.0670288\, x^8 + 0.00615191\, x^9$$

Plot the Euler equation errors normalized by u'[css]

In[1538]:=
```
normalizedError = (Eulerf[x] / u'[css]) /. sola;
Plot[normalizedError, {x, xmin, xmax}]
```

Out[1539]=

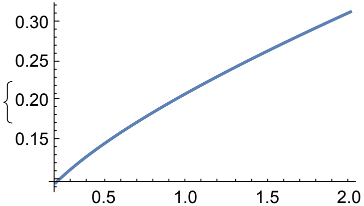


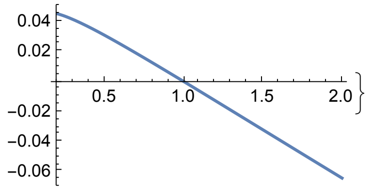
# Summary

Plot consumption and savings functions

```
In[1540]:= {Plot[csol[x], {x, xmin, xmax}], Plot[f[x] - x - csol[x], {x, xmin, xmax}]}
```

Out[1540]=




Lessons

Should use constrained optimization

Must worry about shape

Must keep optimization problems well-conditioned