IMA 2024 Julia Microsimulation Demo

Build a tax-benefit model from scratch in 30 minutes

Things we need:

- data - Family Resources Survey(FRS)/LFS/Living Costs and Food Survey(LCF)/Understanding Society... This example uses our old LCF
- A programming language:
  - We use Julia
- Structures: we need to model:
  - people, families and households;
  - the fiscal system (taxes, benefits);
  - outcomes: incomes net of taxes and benefits, revenues raised, inequalities, poverty and so on.

Let's build one from scratch, in real time (with some 'here's some I prepared earlier' where needs be).

Table of Contents

Load the Libraries we need.

```
begin
 using DataFrames,CSV,Downloads,Plots,StatsPlots, Formatting,
PovertyAndInequalityMeasures, PlutoUI
end
```

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

Load Our Old Living Costs and Food (LCF) Data (Again)

Each row represents a single household in the UKDS 2005/6 LCF teaching dataset. I've simplified the data a bit and made the names a bit clearer.

```
md"""
## Load Our Old Living Costs and Food (LCF) Data (Again)

Each row represents a single household in the [UKDS 2005/6 LCF teaching dataset]
(https://beta.ukdataservice.ac.uk/datacatalogue/studies/study?id=6117). I've
simplified the data a bit and made the names a bit clearer.

"""
```

	hhsize	children	region	tenure	hhincome	gross_pay	nur
1	1	0	"North East"	"Owned outright"	279.95	0.0	0
2	3	1	"North East"	"Mortgaged"	463.851	268.43	2
3	1	0	"North East"	"Other"	40.0	0.0	0
4	2	0	"North East"	"Rented"	285.462	0.0	1
5	4	2	"North East"	"Mortgaged"	610.47	543.5	3
6	1	0	"North East"	"Mortgaged"	27.18	34.9	1
7	3	1	"North East"	"Mortgaged"	639.37	657.45	2
8	2	0	"North East"	"Rented"	445.0	396.44	2
9	1	0	"North East"	"Rented"	91.92	0.0	0
10	4	0	"North East"	"Rented"	768.17	481.39	3
more							
6785	2	1	"Northern Ireland"	"Rented"	120.0	0.0	0

```
begin
url="https://virtual-worlds.scot/ou/uk-lcf-subset-2005-6.csv"

# load LCF into a DataFrame (a spreadsheet-like structure, like Python Pandas, R Tibble)
lcf = CSV.File(Downloads.download(url))|>DataFrame
end
```

Our Tax-Benefit System

This is deliberately very simple. In the first instance, we want to model how a single household is affected by one tax-benefit system. We then build up from that.

We need:

- something to describe the tax-benefit system;
- something to hold results;
- a description of a single household; and
- a function that takes a single household and a system and returns a result

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

```
begin

    # holder for results from one household
    mutable struct Result
        net :: Number
        tax :: Number
        ni  :: Number
        vat :: Number
        benefit :: Number
    end

    # holder for 1 tax system
    struct System
        allowance    :: Number
        it_rate      :: Number
        ni_rate      :: Number
        benefit      :: Number
        vat_rate     :: Number
        extend_vat   :: Bool
    end

    # calculate for one household and one system, returning one result
    function calc_system( hh::DataFrameRow, sys :: System ) :: Result
        out = Result(0.0,0.0,0.0,0.0,0.0)
        taxable = max(0.0, hh.hhincome-sys.allowance )
        out.tax = taxable*sys.it_rate
        out.ni = max(0.0, hh.gross_pay*sys.ni_rate)
        out.benefit = 0.0
        # an unemployment/retirement benefit
        if hh.number_in_work == 0
            out.benefit = sys.benefit
        end
        vatable =
            hh.tobacco_alcohol +
            hh.clothing +
            hh.housing_and_energy +
            hh.household_goods +
            hh.health +
            hh.transport +
            hh.communication +
            hh.recreation +
            hh.restaurants_etc +
            hh.misc_goods
        if sys.extend_vat
            vatable += hh.food_and_drink + hh.housing_and_energy+hh.education
        end
        out.vat = vatable * sys.vat_rate
        out.net = hh.hhincome - out.ni - out.tax - out.vat + out.benefit
        return out
    end

end
```

Analyse And Display The Results

At the other end of the process, we'll have lists of results, one per household, possibly for 2 or more of our tax systems. We'll need to summarise and compare them. The next two functions do this. There's some messy detail here, especially for the charts. We use the `PoveryAndInequalityMeasures` package to produce deciles and Lorenz curves.

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

```
function display(res)
    md"""
    ### Tax revenue
    before: **$(res.tax1)** after: **$(res.tax2)** change: **$(res.dtax)** £mn pa

    ### Benefit Spending
    before: **$(res.ben1)** after: **$(res.ben2)** change: **$(res.dben)** £m pa

    ### Inequality
    Gini before: **$(res.gini1)** after: **$(res.gini2)** change: **$(res.dgini)**

    Palma before: **$(res.palma1)** after: **$(res.palma2)** change:
    **$(res.dpalma)**

    ### Gainers & Losers
    Households gaining: **$(res.gainers)** losing: **$(res.losers)** unchanged:
    **$(res.nc)**
    """
end
```

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

```
begin

    # res1 and res2 are DataFrames holding lists of results for a pre- and post-system run. See the next cells for more. Return a `NamedTuple` of summary results.
    function analyse( res1::DataFrame, res2::DataFrame )::NamedTuple
        tax1 = sum(res1.tax .+ res1.ni .+ res1.vat)*52*WEIGHT
        tax2 = sum(res2.tax .+ res2.ni .+ res2.vat)*52*WEIGHT
        ben1 = sum(res1.benefit)*52*WEIGHT
        ben2 = sum(res2.benefit)*52*WEIGHT
        dben = ben2-ben1
        dtax = tax2-tax1
        ineq1 = make_inequality( res1, :weight, :net )
        ineq2 = make_inequality( res2, :weight, :net )
        dgini = ineq2.gini - ineq1.gini
        dpalma = ineq2.palma - ineq1.palma

        gainers = sum(res1[res2.net .> res1.net,:weight])
        losers = sum(res1[res2.net .< res1.net,:weight])
        nc = sum(res1[res2.net .== res1.net,:weight])

        gain = ineq2.deciles[:,4] .- ineq1.deciles[:,4]
        deciles = bar( string.(1:10), gain, xlabel="Decile", ylabel="£s pw", title="Gains/Losses By Decile", labels="Av Change £s pw" )

        i1 = copy(ineq1.deciles[:,1:2])
        i1 = vcat([0.0 0.0],i1)
        i2 = copy(ineq2.deciles[:,1:2])
        i2= vcat([0.0 0.0],i2)
        lorenz = plot(i1[:,1],i1[:,2],
            labels="res1",
            title="Lorenz Curve",
            xlabel="population share",
            ylabel="Income Share",
            xrange=(0,1),
            yrange=(0,1))
        plot!(lorenz,i2[:,1],i2[:,2],labels="res2" )
        plot!(lorenz, 0:1, 0:1,labels="Equality" )

        (;
         lorenz = lorenz,
         deciles = deciles,
         ben1 = fm( ben1 ),
         ben2 = fm( ben2 ),
         dben = fm(dben),
         gini1=fp(ineq1.gini),
         gini2=fp(ineq2.gini),
         palma1=fp(ineq1.palma),
         palma2=fp(ineq2.palma),
         dpalma=fp(dpalma),
         dgini=fp(dgini),
         tax1=fm( tax1 ),
         tax2=fm( tax2 ),
         dtax=fm( dtax ),
         gainers=fc( gainers ),
         losers=fc( losers ),
         nc=fc(nc))
```

```
    end
end
```

Initialise Systems

These construct our two tax-benefit systems. So `sys1` has a 10k tax allowance, and so on. Sys2, the changed system, is set from a bunch of variables we set below: `new_allowance` and so on. This allows us to change the parameters dynamically.

```
System(192.30769230769232, 0.25, 0.1, 100, 0.2, false)
```

```
begin
    sys1 = System( 10_000/52, 0.25, 0.10, 100.0, 0.20, false )
    sys2 = System( new_allowance/52, new_it_rate/100, new_ni_rate/100, new_benefit,
    new_vat_rate/100, new_extend_vat )
end
```

Sample Weights

Some very crude stuff to uprate and weight our very old LCF data.

- We use a single household weight which is just the number of households in the UK from the ONS divided by the number of sample housholds.
- We uprate by the change in Nominal GDP between 2005 Q3 and 2023 Q3. Note the slightly strange syntax for the multiplication: `!` means 'select all rows', then there's a list of all the columns we need to uprate, then `.*= INFLATION` is a `broadcast multiply` operation.

```
""
```

```
begin
    const SAMPLE_SIZE = size(lcf)[1] # count of rows
    const NUM_HHLDS = 24_783_192
    const WEIGHT = NUM_HHLDS/SAMPLE_SIZE
    # https://www.ons.gov.uk/economy/grossdomesticproductgdp/timeseries/abmz/ukea
    const INFLATION = 678129/353999
    ## nom gdp q3 2004 332192 q3 2023 678129
    lcf[!,
        [:hhincome,
            :gross_pay,
            :food_and_drink,
            :tobacco_alcohol,
            :clothing,
            :housing_and_energy,
            :household_goods,
            :health,
            :transport,
            :communication,
            :recreation,
            :education,
            :restaurants_etc,
            :misc_goods,
            :total_consumption,
            :non_consumption,
            :total_expenditure]] .*= INFLATION
    ""
end
```

Run the model over each household

Next, run our single household/single system function for each household in the data. To save the results, we construct a new DataFrame to hold vectors of tax payments, etc., and fill that in as we go along. We run this function twice, once foreach system, and pass the result DataFrames to the analysis function, which summarises everything for us.

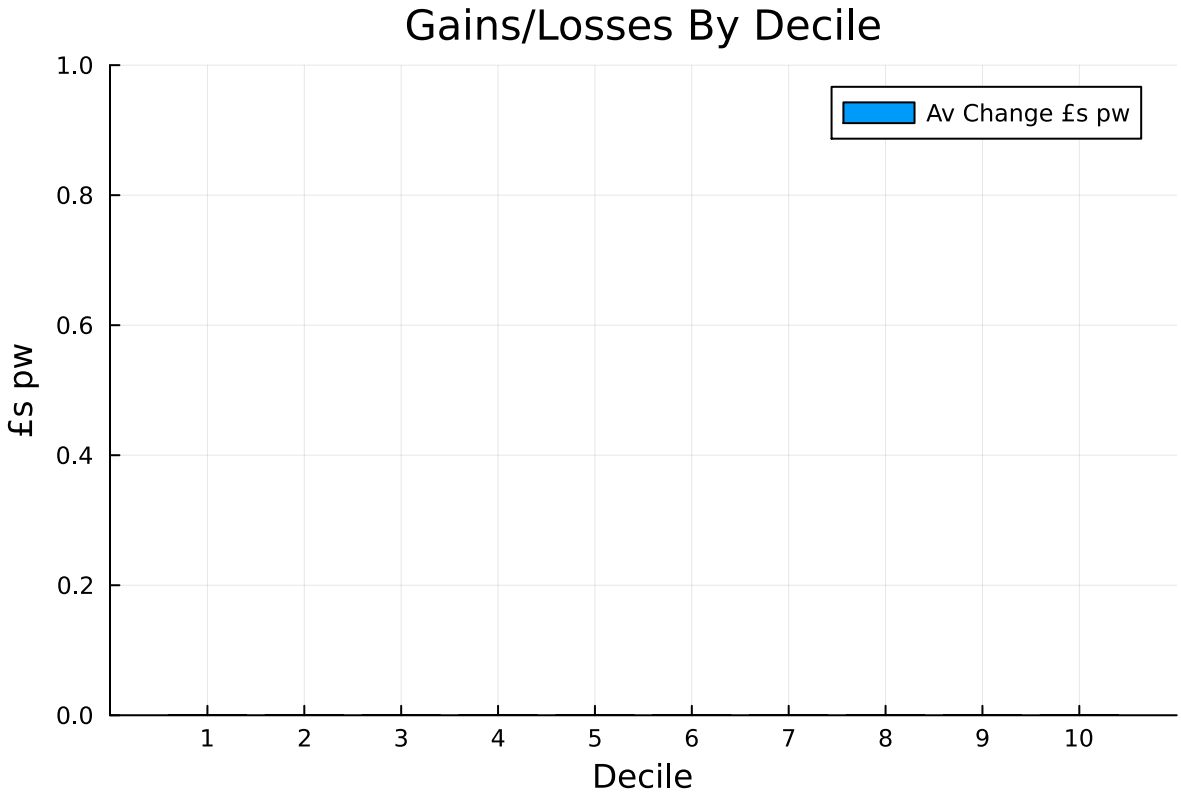
```
""
```

```
begin
    function one_calc( sys :: System ) :: DataFrame
        n = size(lcf)[1]
        out = DataFrame( gross=zeros(n), net=zeros(n), tax=zeros(n), ni=zeros(n),
        vat=zeros(n), benefit=zeros(n) )
        i = 0
        for hh in eachrow( lcf )
            i += 1
            res = calc_system( hh, sys )
            out.gross[i] = hh.hhincome
            out.net[i] = res.net
            out.tax[i] = res.tax
            out.ni[i] = res.ni
            out.vat[i] = res.vat
            out.benefit[i] = res.benefit
        end
        out.weight = fill(WEIGHT,n)
        return out
    end

    res1 = one_calc( sys1 )
    res2 = one_calc( sys2 )
    res = analyse( res1, res2 )
    ""
end
```

Model Inputs and Outputs

Finally, we bring all this together in an interactive microsimulation model. We display the graphs and summary output created above, and below that we bind each of our system 2 parameters (new tax allowance and so on) to simple input fields. Each time you change one of these inputs, the model will react instantly and recalculate everything. It's fast!



Tax revenue

before: **£457,911mn** after: **£457,911mn** change: **£0mn** £mn pa

Benefit Spending

before: **£44,312mn** after: **£44,312mn** change: **£0mn** £m pa

Inequality

Gini before: **34.5%** after: **34.5%** change: **0.0%**

Palma before: **138.3%** after: **138.3%** change: **0.0%**

Gainers & Losers

Households gaining: **0** losing: **0** unchanged: **24,783,192**

tax allowance: 10000 (p.a.)

income tax rate: 25 (%)

vat: 20 (%)

extend vat? ☐

payroll tax: 10 (%)

benefit rate 100 p.w.

```
md"""
tax allowance: $(@bind new_allowance NumberField(0:200:25000,default=10000)) (p.a.)

income tax rate: $(@bind new_it_rate NumberField(0:1:75,default=25)) (%)

vat: $(@bind new_vat_rate NumberField(0:1:75,default=20)) (%)

extend vat? $(@bind new_extend_vat CheckBox(default=false))

payroll tax: $(@bind new_ni_rate NumberField(0:1:75,default=10)) (%)

benefit rate $(@bind new_benefit NumberField(0:5:250,default=100)) p.w.

"""
```

tax allowance pre: **£10,000p.a** post: **£10,000p.a**

income tax rate: pre: **25%** post: **25%**

vat: pre: **20%** post: **20%**

extend vat? pre: **false** post: **false**

payroll tax: pre: **10%** post: **10%**

benefit rate pre: **£100p.w.** post **£100p.w.**