Angrist 1998: Regression Weights

Source: `refs/angrist1988.pdf`, Joshua D. Angrist, "Estimating the Labor Market Impact of Voluntary Military Service Using Social Security Data on Military Applicants," Econometrica, 1998. The local filename says `1988`, but the paper is the 1998 Econometrica article.

Sections 2.1 and 2.2. Section 2.1 sets up potential outcomes Y_0 and Y_1, treatment D, and covariates X. The target is the effect of treatment on the treated. Under conditional ignorability, $(Y_1, Y_0) \perp D \mid X$, the treated effect can be written as a weighted average of within-covariate treatment-control contrasts. The matching estimand weights each covariate-specific contrast by the distribution of X among treated units.

Section 2.2 compares that matching estimand with the coefficient on D in a regression of Y on D and controls for X. With heterogeneous treatment effects, the regression coefficient is also a weighted average of covariate-specific effects, but it uses overlap weights. In the saturated-control case, the weights are proportional to

$$P(D = 1 \mid X)(1 - P(D = 1 \mid X))P(X).$$

Thus cells with no treatment variation get zero weight, and cells where treatment is close to half treated receive relatively more weight than they do under treated-distribution matching.

Algebraic Claim. The deterministic algebra behind Section 2.2 has two steps. First, by the Frisch-Waugh-Lovell theorem, the treatment coefficient from the full regression equals the coefficient from regressing residualized outcomes on residualized treatment. Let `controls` be the control design matrix, let `d` be the treatment vector, let `y` be the outcome vector, and define the residual-maker

$$M_X = I - X(X'X)^{-1}X'.$$

If the relevant Gram matrices are invertible, the treatment coefficient from the full regression on `[X D]` equals the coefficient from the auxiliary regression of residualized outcomes on residualized treatment:

$$\hat{\alpha} = \text{coef}_D(Y \sim X, D) = \text{coef}(M_X Y \sim M_X D) = (D' M_X D)^{-1} D' M_X Y.$$

Second, when `controls` are saturated covariate-cell indicators, expanding $M_X D$ gives within-cell treatment residuals $D_i - p_x$, where $p_x = P(D = 1 \mid X = x)$. If cell x has mass m_x, treated mean outcome y_{1x}, and untreated mean outcome y_{0x}, then the numerator and denominator become

$$\sum_x m_x p_x (1 - p_x)(y_{1x} - y_{0x}) \quad \text{and} \quad \sum_x m_x p_x (1 - p_x).$$

Therefore the saturated-regression coefficient is the overlap-weighted average

$$\frac{\sum_x m_x p_x (1 - p_x)(y_{1x} - y_{0x})}{\sum_x m_x p_x (1 - p_x)}.$$

The causal interpretation still requires the potential-outcomes assumptions from Section 2.1. The Lean result proved here is the finite-cell regression algebra that produces Angrist's overlap weights.

Lean Result. The application module is `applications/Angrist1998.lean`.

Key declarations:

- `treatmentDesign`: represents a scalar treatment vector as a one-column matrix.
- `regressionCoefficient`: the treatment coefficient from the full regression on controls and treatment.
- `residualizedCoefficient`: the FWL auxiliary coefficient from residualized treatment and residualized outcomes.
- `regressionCoefficient_eq_residualizedCoefficient`: proves the FWL identity specialized to Angrist's regression-weight setup.
- `regressionCoefficient_eq_partitionedFormula`: proves the displayed partitioned formula `(D' M_X D)^{-1} D' M_X Y` in the existing Hansen matrix notation.
- `cellResidualizedTreatmentOutcomeMoment_eq_overlap_sum`: proves the saturated-cell numerator is `sum_x m_x p_x (1-p_x) (y1_x-y0_x)`.
- `cellResidualizedTreatmentSecondMoment_eq_overlap_sum`: proves the saturated-cell denominator is `sum_x m_x p_x (1-p_x)`.
- `cellRegressionCoefficient_eq_overlapWeightedTreatmentEffect`: proves the cell-level regression coefficient is the overlap-weighted treatment-effect average.

Lean Proof Walkthrough. The proof worked: `applications/Angrist1998.lean` compiles, and the top-level `lake build` now builds the `applications` library as well as the core `HansenEconometrics` library.

The application module deliberately proves only a thin, Angrist-shaped wrapper around the existing Hansen Chapter 3 FWL machinery. The reason is that Section 2.2's regression-weight claim starts from a standard regression coefficient. Before expanding that coefficient into cell weights, the first algebraic step is simply the FWL identity:

$$\mathrm{coef}_D(Y \sim X, D) = \mathrm{coef}(M_X Y \sim M_X D).$$

The module starts with:

```
import HansenEconometrics.Chapter3FWL
```

This import supplies the existing finite-sample regression algebra:

- `Matrix.fromCols X1 X2`: the block design `[X1 X2]`.
- `fromColsRightBeta X1 X2 y`: the right-block coefficient from the full regression.
- `residualizedRegressors X1 X2`: the residualized right block `M1 X2`.
- `fwlBeta X1 X2 y`: the auxiliary coefficient from regressing `M1 y` on `M1 X2`.
- `partitionedRightBetaFormula X1 X2 y`: the explicit `(X2' M1 X2)^{-1} X2' M1 y` formula.
- `fromColsRightBeta_eq_fwlBeta`: Hansen's FWL coefficient theorem.
- `fromColsRightBeta_eq_partitionedRightBetaFormula`: Hansen's explicit partitioned-regression formula.

The application-level row type is `n`, and the control-column type is `c`:

```
variable {n c : Type*}
variable [Fintype n]
```

The rows must be finite because all regression objects are finite matrices. The control type `c` is not declared finite globally; instead, the definitions and theorems request `[Fintype c] [DecidableEq c]` exactly where matrix multiplication and block-column indexing need them.

Angrist's treatment variable is scalar, while the Hansen FWL theorem expects a matrix block. The bridge is a one-column design:

```
noncomputable def treatmentDesign (d : n → ℝ) : Matrix n Unit ℝ :=
  fun i _ => d i
```

The column index is `Unit`, which has exactly one element. Thus a vector-valued coefficient indexed by `Unit` is a scalar coefficient in disguise. Evaluating that coefficient at `()` extracts the scalar.

The full-regression treatment coefficient is:

```
noncomputable def regressionCoefficient
    (controls : Matrix n c ℝ) (d y : n → ℝ)
    [Fintype c] [DecidableEq c]
    [Invertible ((Matrix.fromCols controls (treatmentDesign d))ᵀ *
      Matrix.fromCols controls (treatmentDesign d))] : ℝ :=
  fromColsRightBeta controls (treatmentDesign d) y ()
```

Mathematically, this is the coefficient on `D` in the regression of `Y` on `[X D]`. The invertibility assumption is the usual full-rank Gram condition for the full design. The expression `fromColsRightBeta controls (treatmentDesign d) y` has type `Unit → ℝ`, so applying it to `()` returns the one treatment coefficient.

The residualized coefficient is:

```
noncomputable def residualizedCoefficient
    (controls : Matrix n c ℝ) (d y : n → ℝ)
    [DecidableEq n] [Fintype c] [DecidableEq c]
    [Invertible (controlsᵀ * controls)]
    [Invertible ((residualizedRegressors controls (treatmentDesign d))ᵀ *
      residualizedRegressors controls (treatmentDesign d))] : ℝ :=
  fwlBeta controls (treatmentDesign d) y ()
```

This is the coefficient from regressing `M_XY` on `M_XD`. The extra assumptions say:

- `controlsᵀ * controls` is invertible, so the control residual-maker `M_X` is defined.
- `(M_XD)'(M_XD)` is invertible, so the auxiliary one-regressor regression is defined.
- `DecidableEq n` is needed because the annihilator matrix uses the identity matrix on rows.

The main theorem is:

```
theorem regressionCoefficient_eq_residualizedCoefficient
    (controls : Matrix n c ℝ) (d y : n → ℝ)
    [DecidableEq n] [Fintype c] [DecidableEq c]
    [Invertible (controlsᵀ * controls)]
    [Invertible ((Matrix.fromCols controls (treatmentDesign d))ᵀ *
      Matrix.fromCols controls (treatmentDesign d))]
    [Invertible ((residualizedRegressors controls (treatmentDesign d))ᵀ *
      residualizedRegressors controls (treatmentDesign d))] :
    regressionCoefficient controls d y = residualizedCoefficient controls d y := by
  unfold regressionCoefficient residualizedCoefficient
  exact congrFun (fromColsRightBeta_eq_fwlBeta controls (treatmentDesign d) y) ()
```

The proof has two steps.

First, `unfold regressionCoefficient residualizedCoefficient` replaces the two application-level names by their definitions. The goal becomes:

```
fromColsRightBeta controls (treatmentDesign d) y ()
  = fwlBeta controls (treatmentDesign d) y ()
```

Second, the existing Chapter 3 theorem

```
fromColsRightBeta_eq_fwlBeta controls (treatmentDesign d) y
```

proves equality of the full `Unit → ℝ` coefficient vectors:

```
fromColsRightBeta controls (treatmentDesign d) y
  = fwlBeta controls (treatmentDesign d) y
```

Since this is equality of functions, `congrFun ... ()` applies both sides to the unique `Unit` value `()`. That extracts the scalar treatment coefficient and closes the goal.

The second theorem records the explicit display formula:

```
theorem regressionCoefficient_eq_partitionedFormula
    ...
    regressionCoefficient controls d y =
      partitionedRightBetaFormula controls (treatmentDesign d) y () := by
  unfold regressionCoefficient
  exact congrFun (fromColsRightBeta_eq_partitionedRightBetaFormula
    controls (treatmentDesign d) y) ()
```

This proof is identical in structure. After unfolding `regressionCoefficient`, the goal is the scalar version of the already-proved partitioned-regression theorem. The theorem `fromColsRightBeta_eq_partitionedRightBetaFormula` gives the vector identity, and `congrFun ... ()` extracts the unique scalar coordinate. In mathematical notation, the right side is:

$$(D'M_XD)^{-1}D'M_XY.$$

The FWL result is only the first layer. The second layer adds a finite cell type `g`:

```
variable {g : Type*}
variable [Fintype g]
```

Here `g` indexes saturated covariate cells. At this level we no longer need individual row identifiers. Each cell carries:

- `cellMass x`: the mass or population share of cell `x`;
- `propensity x`: the treatment probability $p_x = P(D = 1 \mid X = x)$;
- `y0 x`: the untreated cell mean outcome;
- `y1 x`: the treated cell mean outcome.

Treatment status is represented by `Bool`. The real-valued treatment indicator is:

```
def binaryTreatmentValue : Bool → ℝ
  | false => 0
  | true => 1
```

The cell joint mass for a cell and treatment state is:

```
def cellJointMass (cellMass propensity : g → ℝ) (x : g) (d : Bool) : ℝ :=
  cellMass x * if d then propensity x else 1 - propensity x
```

So treated observations in cell `x` receive mass `m_x p_x`, and untreated observations receive mass `m_x(1-p_x)`. The outcome mean and residualized treatment are:

```
def cellOutcomeMean (y0 y1 : g → ℝ) (x : g) (d : Bool) : ℝ :=
  if d then y1 x else y0 x

def cellTreatmentResidual (propensity : g → ℝ) (x : g) (d : Bool) : ℝ :=
  binaryTreatmentValue d - propensity x
```

Thus the treated residual is `1 - p_x`, and the untreated residual is `-p_x`.

The overlap weight and cell treatment contrast are named directly:

```
def overlapWeight (cellMass propensity : g → ℝ) (x : g) : ℝ :=
  cellMass x * propensity x * (1 - propensity x)

def cellTreatmentEffect (y0 y1 : g → ℝ) (x : g) : ℝ :=
  y1 x - y0 x
```

The numerator of the residualized-treatment regression is formalized as the joint cell-by-treatment sum

$$\sum_x \sum_d m_x P(D = d \mid X = x)(d - p_x)E[Y \mid X = x, D = d].$$

In Lean:

```
noncomputable def cellResidualizedTreatmentOutcomeMoment
    (cellMass propensity y0 y1 : g → ℝ) : ℝ :=
  ∑ x : g, ∑ d : Bool,
    cellJointMass cellMass propensity x d *
      cellTreatmentResidual propensity x d *
        cellOutcomeMean y0 y1 x d
```

The denominator is the residualized-treatment second moment:

```
noncomputable def cellResidualizedTreatmentSecondMoment
    (cellMass propensity : g → ℝ) : ℝ :=
  ∑ x : g, ∑ d : Bool,
    cellJointMass cellMass propensity x d *
      (cellTreatmentResidual propensity x d) ^ 2
```

The first real Angrist-weight theorem expands the numerator:

```
theorem cellResidualizedTreatmentOutcomeMoment_eq_overlap_sum
    (cellMass propensity y0 y1 : g → ℝ) :
    cellResidualizedTreatmentOutcomeMoment cellMass propensity y0 y1 =
      ∑ x : g, overlapWeight cellMass propensity x * cellTreatmentEffect y0 y1 x := by
  unfold cellResidualizedTreatmentOutcomeMoment overlapWeight cellTreatmentEffect
  refine Finset.sum_congr rfl ?_
  intro x _
  simp [cellJointMass, cellTreatmentResidual, binaryTreatmentValue, cellOutcomeMean]
  ring
```

After unfolding, Lean proves the identity cell-by-cell. For a fixed cell, the `Bool` sum has two terms:

$$\begin{aligned} &m_x p_x(1 - p_x)y_{1x} + m_x(1 - p_x)(-p_x)y_{0x} \\ &= m_x p_x(1 - p_x)(y_{1x} - y_{0x}). \end{aligned}$$

The `simp` step evaluates the `Bool` cases; `ring` closes the polynomial identity.

The denominator theorem is analogous:

```
theorem cellResidualizedTreatmentSecondMoment_eq_overlap_sum
    (cellMass propensity : g → ℝ) :
    cellResidualizedTreatmentSecondMoment cellMass propensity =
      ∑ x : g, overlapWeight cellMass propensity x := by
  unfold cellResidualizedTreatmentSecondMoment overlapWeight
  refine Finset.sum_congr rfl ?_
  intro x _
  simp [cellJointMass, cellTreatmentResidual, binaryTreatmentValue]
  ring
```

For each cell, it proves

$$m_x p_x (1 - p_x)^2 + m_x (1 - p_x) p_x^2 = m_x p_x (1 - p_x).$$

The final theorem defines the ratio of these two residualized moments as `cellRegressionCoefficient`, defines the overlap-weighted estimand as `overlapWeightedTreatmentEffect`, and rewrites both numerator and denominator:

```
theorem cellRegressionCoefficient_eq_overlapWeightedTreatmentEffect
    (cellMass propensity y0 y1 : g → ℝ) :
    cellRegressionCoefficient cellMass propensity y0 y1 =
      overlapWeightedTreatmentEffect cellMass propensity y0 y1 := by
  unfold cellRegressionCoefficient overlapWeightedTreatmentEffect
  rw [cellResidualizedTreatmentOutcomeMoment_eq_overlap_sum,
    cellResidualizedTreatmentSecondMoment_eq_overlap_sum]
```

This is the finite-cell version of Angrist's Section 2.2 expression:

$$\frac{\sum_x m_x p_x (1 - p_x)(y_{1x} - y_{0x})}{\sum_x m_x p_x (1 - p_x)}.$$

The module also records that overlap weights are nonnegative under $m_x \geq 0$ and $0 \leq p_x \leq 1$, and that cells with $p_x = 0$ or $p_x = 1$ receive zero weight. That matches Angrist's point that regression drops cells with no within-cell treatment variation.

The next natural applications are `refs/regrank.pdf`, which studies regression-induced ranking reversals, and `refs/2106.05024v5.pdf`, Goldsmith-Pinkham, Hull, and Kolesar's "Contamination Bias in Linear Regressions."