Formalization of Bilinear Forms Proofs

Clea Bergsman

2025-08-06

Outline

Bilinear Forms

Definition

A **bilinear form** is a map $\beta : V \times W \to K$, where V and W are K-vector spaces and K is a field, when

1. $\beta(v_1 + v_2, w) = \beta(v_1, w) + \beta(v_2, w)$
2. $\beta(v, w_1 + w_2) = \beta(v, w_1) + \beta(v, w_2)$
3. $\beta(\lambda v, w) = \beta(v, \lambda w) = \lambda\beta(v, w)$

hold $\forall\ v \in V$, $w \in W$, and $\lambda \in K$.

Equivalent Bilinear Forms

Definition

Two bilinear forms β_1 and β_2 on the respective vector spaces V_1 and V_2 are **equivalent** if there is a vector space isomorphism $\Phi : V_1 \to V_2$ such that $\beta_2(\Phi v, \Phi w) = \beta_1(v, w) \ \forall \ v, w \in V_1$

Proof Statement

- Given two bilinear forms, β_1 and β_2, on the respective vector spaces V_1 and V_2 and a basis b_1 for β_1
- Show that β_1 is equivalent to $\beta_2 \iff \exists$ a basis b_2 of V_2 such that M_1 given by $[\beta_1 \ (b_1 \ i, \ b_1 \ j)]$ is equal to M_2 given by $[\beta_2 \ (b_2 \ i, \ b_2 \ j)]$

Proving Equivalence of Bilinear Forms

Proof Statement $\rightarrow$

Given that β_1 and β_2 are equivalent, show that M_1 is equivalent to M_2.

Steps:

1. Define $\Phi : V_1 \rightarrow V_2$ with two properties:
   - Equivalence: Φ is an invertible linear transformation
   - Compatibility : $\beta_1(v, w) = \beta_2(\Phi v, \Phi w)$
2. Construct b_2 as a basis from b_1 using Φ
   - If $b_1 = x_1, \ldots, x_n$ then $b_2 = \Phi x_1, \ldots, \Phi x_n$
3. $M_1 = M_2$ by compatibility

Proof Statement $\leftarrow$

Given a basis b_2 of V_2 such that $M_1 = M_2$, show that β_1 is equivalent to β_2.

Steps:

1. Define $\Phi : V_1 \rightarrow V_2$ where $\Phi(b_1\ i) = b_2\ i$
2. Check compatibility condition holds
   - Compatibility is true on a basis, so must check that compatibility is true for all vectors
3. Show that all vectors can be written as a linear combination of basis vectors, therefore compatibility holds

Linearity of Sums in Bilinear Forms

Lemma

$$\beta(\sum_i t_i \bullet b_i, \sum_j s_j \bullet b_j) = \sum_i \sum_j t_i * s_j \ \beta(b_i, b_j)$$

Where β is a bilinear form, t_i, $s_j \in k$, and b_i, b_j are basis vectors for V

Recall:

Definition

A **bilinear form** is a map $\beta : V \times W \to K$, where V and W are K-vector spaces and K is a field, when

1. $\beta(v_1 + v_2, w) = \beta(v_1, w) + \beta(v_2, w)$
2. $\beta(v, w_1 + w_2) = \beta(v, w_1) + \beta(v, w_2)$
3. $\beta(\lambda v, w) = \beta(v, \lambda w) = \lambda\beta(v, w)$

hold $\forall$ $v \in V$, $w \in W$, and $\lambda \in K$.

Linearity of Sums in Bilinear Forms Lean Proof

```
lemma equiv_of_series {ι:Type} [Fintype ι]
    (β:BilinForm k V) (b : Basis ι k V) (s t : ι → k):
(β (Fintype.linearCombination k ⇑b t))
(Fintype.linearCombination k ⇑b s) =
∑ i:ι, (∑ j:ι, (t i) * (s j) * (β (b i) (b j))) := by
  unfold Fintype.linearCombination
  dsimp
  rw [LinearMap.BilinForm.sum_left]
  apply Finset.sum_congr
  rfl
  intro i h
  rw [LinearMap.BilinForm.sum_right]
  apply Finset.sum_congr
  rfl
  intro j g
  rw [LinearMap.BilinForm.smul_left]
  rw [mul_comm]
  rw [LinearMap.BilinForm.smul_right]
  ring
```

Proving Equivalence in Lean

```
theorem equiv_via_matrices  {ι:Type} [Fintype ι] [DecidableEq ι]
 (β₁:BilinForm k V₁) (β₂:BilinForm k V₂) (b₁ : Basis ι k V₁) (i j : ι) (s t : ι → k)
  : Nonempty (equiv_of_spaces_with_form β₁ β₂) ↔  ∃ b₂:Basis ι k V₂, ∀ i j : ι,
    (BilinForm.toMatrix b₁ β₁) i j =  (BilinForm.toMatrix b₂ β₂) i j
  := by
  constructor
  -- mp
  intro <N>
  let b₂ : Basis ι k V₂ := Basis.map b₁ N.equiv
  use b₂
  unfold b₂
  unfold BilinForm.toMatrix
  simp
  intro i j
  rw [N.compat (b₁ i) (b₁ j)]
  -- mpr
  intro h₁
  rcases h₁ with <b₂, h₁>
  refine Nonempty.intro ?_
  let eq : V₁ ≃[k] V₂ := by apply equiv_from_bases; exact b₁; exact b₂
  have identify_bases : ∀ i:ι, b₂ i = eq (b₁ i) := by
    intro i; unfold eq;  rw [← equiv_from_bases_apply b₁ b₂ i]
  apply equiv_of_spaces_with_form.mk
  intro v w
  swap
  exact eq
```

Proving Equivalence in Lean Continued

```
have sum_v : v = (Fintype.linearCombination k ⇑b₁) (b₁.repr v):=
    by symm; apply fintype_linear_combination_repr
  have sum_w : w = (Fintype.linearCombination k ⇑b₁) (b₁.repr w):=
    by symm; apply fintype_linear_combination_repr
  nth_rw 1 [sum_v, sum_w]
  rw [equiv_of_series]
  nth_rw 2 [sum_v, sum_w]
  rw [ Fintype.linearCombination_apply, Fintype.linearCombination_apply]
  rw [ map_sum eq, map_sum eq]
  rw [equiv_of_bilin_series]
  apply Finset.sum_congr
  rfl
  intro i hi
  apply Finset.sum_congr
  rfl
  intro j hj
  rw [map_smul eq, map_smul eq]
  rw [LinearMap.BilinForm.smul_left]
  rw [mul_comm]
  rw [LinearMap.BilinForm.smul_right]
  rw [mul_comm]
  rw [← identify_bases, ← identify_bases]
  rw [← BilinForm.toMatrix_apply b₁ β₁ i j,← BilinForm.toMatrix_apply b₂ β₂]
  rw [h₁ i j]
  ring
```

Alternating Equivalence

Definition

A bilinear form β is **alternating** or **anti-symmetric** if

1. $\beta(v, v) = 0, \ \forall \ v \in V$
2. $\beta(v, w) = -\beta(w, v), \ \forall \ v, w \in V$

```
theorem alt_of_equiv (eq : V₁ ≃[k,β₁,β₂] V₂)
 (halt : β₂.IsAlt) : β₁.IsAlt := by
  intro v
  have β₂_alt : (β₂ (eq.equiv v)) (eq.equiv v) = 0 :=
    by apply halt
  have h₁ : (β₁ v) v = (β₂ (eq.equiv v)) (eq.equiv v) :=
    by apply eq.compat
  rw [β₂_alt] at h₁
  exact h₁
```

Symmetric Equivalence

Definition

A bilinear form β is **symmetric** if

- $\beta(v, w) = \beta(w, v)\ \forall\ v, w \in V$

```
theorem symm_of_equiv (eq : V₁ ≃[k,β₁,β₂] V₂)
  (hsymm : β₂.IsSymm): β₁.IsSymm := by
  intro v w
  have β₂_symm : (β₂ (eq.equiv v)) (eq.equiv w) =
    (β₂ (eq.equiv w)) (eq.equiv v) := by apply hsymm
  have h₁ : (β₁ v) w = (β₂ (eq.equiv v)) (eq.equiv w) :=
    by apply eq.compat
  have h₂ : (β₁ v) w = (β₂ (eq.equiv w)) (eq.equiv v) :=
    by rw [β₂_symm] at h₁; exact h₁
  have h₃ : (β₁ w) v = (β₂ (eq.equiv w)) (eq.equiv v) :=
    by apply eq.compat
  rw [← h₂] at h₃
  simp
  symm
  exact h₃
```

Anisotropic Equivalence

Definition

A bilinear form β is **anisotropic** if

- $\forall\ v \in V,\ v \neq 0,\ \beta(v, v) \neq 0$

```
theorem anisotropic_of_equiv (eq : V₁ ≃[k,β₁,β₂] V₂)
  (han : anisotropic β₂) : anisotropic β₁ := by
  intro v hv
  unfold anisotropicVector
  have h₁ : (β₂ (eq.equiv v)) (eq.equiv v) ≠ 0 := by
    apply han;
    exact (LinearEquiv.map_ne_zero_iff eq.equiv).mpr hv
  have h₂ : (β₁ v) v = (β₂ (eq.equiv v)) (eq.equiv v) := by
    apply eq.compat
  rw [← h₂] at h₁
  exact h₁
```

Nondegenerate Equivalence

Nondegenerate Definition

Let β be a bilinear form on V, $M = [\beta(v_i, v_j)]$, and $v_1, ..., v_n$ a basis of V. The following are equivalent:

- $det(M) \neq 0$
- $\forall w \in V \ \beta(v, w) = 0 \implies v = 0$
- $\forall v \in V \ \beta(v, w) = 0 \implies w = 0$

Nondegenerate Equivalence Lean Proof

```
theorem nondeg_of_equiv (eq : V₁ ≃[k,β₁,β₂] V₂) (hnd : β₂.Nondegenerate)
  : β₁.Nondegenerate := by
  intro v h₁
  have h₂ : (∀ (w : V₂), (β₂ (eq.equiv v)) w = 0) → (eq.equiv v) = 0 := by
    exact hnd (eq.equiv v)
  have h₃ : ∀ (n:V₁), (β₁ v) n = (β₂ (eq.equiv v)) (eq.equiv n) := by
    apply eq.compat
  have eq₁ : eq.equiv v = 0 → v = 0 := by
    intro h
    exact (LinearEquiv.map_eq_zero_iff eq.equiv).mp h
  apply eq₁
  apply h₂
  intro w
  let x : V₁ := eq.equiv.invFun w
  have h₅ : eq.equiv x = w := by exact (LinearEquiv.eq_symm_apply eq.equiv).mp rfl
  rw [← h₅]
  have h₄ : ∀ (n : V₁), (β₂ (eq.equiv v)) (eq.equiv n) = 0 := by
    intro n
    rw [← h₃]
    apply h₁
  apply h₄
```

Symmetry of Equivalence

Recall:

Definition

Two bilinear forms β_1 and β_2 on the respective vector spaces V_1 and V_2 are **equivalent** if there is a vector space isomorphism $\Phi : V_1 \to V_2$ such that $\beta_2(\Phi v, \Phi w) = \beta_1(v, w)\ \forall\ v,\ w \in V_1$

Lemma

Given β_1 is equivalent to β_2, there exists a vector space isomorphism $\phi_2 : V_2 \to V_1$ such that $\beta_1(\Phi_2 v, \Phi_2 w) = \beta_2(v, w)\ \forall$ $v,\ w \in V_2$

Proof of Symmetry of Equivalence in Lean

```
def equiv_of_spaces_with_form.symm {β₁ : BilinForm k V₁}
{β₂:BilinForm k V₂} (e:V₁≃[k,β₁,β₂] V₂) : V₂≃[k,β₂,β₁] V₁ where
    equiv := e.equiv.symm
    compat := by
      intro y z
      let v : V₁ := e.equiv.invFun y
      let w : V₁ := e.equiv.invFun z
      have h₁ : (β₁ v) w = (β₂ (e.equiv v)) (e.equiv w) := by
        apply e.compat
      have hv : e.equiv v = y := by
        exact (LinearEquiv.eq_symm_apply e.equiv).mp rfl
      have hw : e.equiv w = z := by
        exact (LinearEquiv.eq_symm_apply e.equiv).mp rfl
      rw [hv, hw] at h₁
      have hy : v = e.equiv.invFun y := by rfl
      have hz : w = e.equiv.invFun z := by rfl
      rw [hy, hz] at h₁
      simp at h₁
      symm at h₁
      exact h₁
```

Transitivity of Equivalence

Lemma

Given bilinear forms β_1, β_2, and β_3 on the respective vector spaces V_1, V_2, and V_3, $\Phi : V_1 \to V_2$ such that $\beta_2(\Phi v, \Phi w) = \beta_1(v, w)$ $\forall$ v, $w \in V_1$, and $\Phi_2 : V_2 \to V_3$ such that $\beta_3(\Phi_2 v, \Phi_2 w) = \beta_2(v, w)$ $\forall$ v, $w \in V_2$.
$\exists$ a vector space isomorphism $\Phi_3 : V_1 \to V_3$ such that $\beta_3(\Phi_3 v, \Phi_3 w) = \beta_1(v, w)$ $\forall$ v, $w \in V_1$.

Proof of Transitivity of Equivalence in Lean

```
def equiv_of_spaces_with_form.trans  {β₁:BilinForm k V₁}
  {β₂:BilinForm k V₂} {β₃:BilinForm k V₃}
  (e₁:V₁ ≃[k,β₁,β₂] V₂) (e₂:V₂ ≃[k,β₂,β₃] V₃) :
  V₁ ≃[k,β₁,β₃] V₃ where
    equiv := e₁.equiv.trans e₂.equiv
    compat := by
      intro x y
      have h₁ : (β₁ x) y = (β₂ (e₁.equiv x)) (e₁.equiv y) :=
        by apply e₁.compat
      have h₂ : (β₂ (e₁.equiv x)) (e₁.equiv y) =
        (β₃ (e₂.equiv (e₁.equiv x))) (e₂.equiv (e₁.equiv y)) :=
            by apply e₂.compat
      rw [h₂] at h₁
      simp
      exact h₁
```

Lean Takeaways

- Lean is not necessarily an efficient way to prove things given the specificity of types and the time it takes to find necessary theorems and lemmas in Mathlib (or prove them yourself if you can't find it)
  - However, dependent type theory does ensure accuracy in proofs and prevents incorrect inferences
- There are benefits to proving things in Lean, although they do not always outweigh the amount time necessary to spend on proofs
  - A strong understanding of traditional proofs
  - Lean is like any math problem; it can be frustrating but is rewarding to complete
- Lean will become more practical over time as the time spent working on proofs may decrease in the future as Mathlib expands

References

1. Conrad, K. (n.d.). Bilinear Forms. https://kconrad.math.uconn.edu/blurbs/linmultialg/bilinearform.pdf
2. Reich, E. (2005, February 28). Bilinear Forms. Retrieved July 10, 2005, from https://math.mit.edu/ dav/bilinearforms.pdf

Thank you!

This work would not have been possible without the support of the National Science Foundation, the Tufts Math Department, and, most importantly, George McNinch.