線形代数

Jacques Garrigue, 2019 年 7 月 24 日

1 Mathcomp で線形代数

Mathcomp の algebra フォルダが代数学関係のライブラリを与えている。

`zmodp.v`	$\mathbf{Z}/p\mathbf{Z}$
`ssralg.v`	環など
`poly.v`	多項式
`ssrnum.v`	体など
`matrix.v`	行列
`vector.v`	ベクトル空間

2 ベクトル空間

以下の問題を解きます。

1. E を K 上のベクトル空間とする。以下が同値であることを証明せよ。

$$E = \operatorname{Im} f \oplus \operatorname{Ker} f \quad \Leftrightarrow \quad \operatorname{Im} f = \operatorname{Im}(f \circ f)$$

2. E を K 上のベクトル空間とする。p と q を E 上の射影写像とする。
   (a) $p \circ q = q \circ p = 0$ が $p + q$ が射影写像である必要十分条件であることを証明せよ
   (b) $p + q$ が射影写像なら、以下が成り立つことを証明せよ

$$\operatorname{Im}(p+q) = \operatorname{Im} p \oplus \operatorname{Im} q$$

$$\operatorname{Ker}(p+q) = \operatorname{Ker} p \cap \operatorname{Ker} q$$

定義と方法

```
From mathcomp Require Import all_ssreflect all_algebra. (* 代数ライブラリ *)

Local Open Scope ring_scope. (* 環構造を使う *)
Import GRing.Theory.

Section Problem1.

Variable K : fieldType.                                          (* 体 *)
Variable E : vectType K.                          (* 有限次元ベクトル空間 *)
Variable f : 'End(E).                                        (* 線形変換 *)

Theorem equiv1 :
  (limg f + lker f)%VS = fullv <-> limg f = limg (f \o f).
Proof.
split.
+ move/(f_equal (lfun_img f)).
```

```
  rewrite limg_comp limg_add.
  admit.
+ rewrite limg_comp => Hf'.
  move: (limg_ker_dim f (limg f)).
  rewrite -[RHS]add0n -Hf' => /eqP.
  rewrite eqn_add2r dimv_eq0 => /eqP /dimv_disjoint_sum.
Admitted.

End Problem1.


Section Problem2.

Variable K : numFieldType.                              (* ノルム付き体 *)
Variable E : vectType K.
Variable p q : 'End(E).

Definition projection (f : 'End(E)) := forall x, f (f x) = f x.

Lemma proj_idE f : projection f <-> {in limg f, f =1 id}.
Proof.
split => Hf x.
+ by move/limg_lfunVK => <-.
+ by rewrite Hf // memv_img ?memvf.
Qed.

Hypothesis proj_p : projection p.
Hypothesis proj_q : projection q.

Section a.

Lemma f_g_0 f g x :
  projection f -> projection g -> projection (f+g) -> f (g x) = 0.
Proof.
move=> Pf Pg /(_ (g x)).
rewrite !add_lfunE !linearD /=.
rewrite !Pf !Pg => /eqP.
rewrite -subr_eq !addrA addrK.
rewrite addrAC eq_sym -subr_eq eq_sym subrr => /eqP Hfg.
move: (f_equal g Hfg).
rewrite !linearD /= Pg linear0 => /eqP.
Admitted.

Theorem equiv2 :
  projection (p + q) <-> (forall x, p (q x) = 0 /\ q (p x) = 0).
Proof.
split=> H x.
Admitted.

End a.
```

```
Section b.

Hypothesis proj_pq : projection (p + q).

Lemma b1a x : x \in limg p -> x \in limg q -> x = 0.
Admitted.

Lemma b1b : directv (limg p + limg q).
Proof.
apply/directv_addP/eqP.
rewrite -subv0.
apply/subvP => u /memv_capP [Hp Hq].
rewrite memv0.
Admitted.

Lemma limg_sub_lker f g :
  projection f -> projection g -> projection (f+g) -> (limg f <= lker g)%VS.
Admitted.

Lemma b1c : (limg p <= lker q)%VS.
Admitted.

Lemma b1c' : (limg q <= lker p)%VS.
Admitted.

Lemma limg_addv (f g : 'End(E)) : (limg (f + g)%R <= limg f + limg g)%VS.
Proof.
apply/subvP => x /memv_imgP [u _ ->].
Admitted.

Theorem b1 : limg (p+q) = (limg p + limg q)%VS.
Proof.
apply/eqP; rewrite eqEsubv limg_addv /=.
apply/subvP => x /memv_addP [u Hu] [v Hv ->].
have -> : u + v = (p + q) (u + v).
  rewrite lfun_simp !linearD /=.
  rewrite (proj1 (proj_idE p)) // (proj1 (proj_idE q) _ v) //.
Admitted.

Theorem b2 : lker (p+q) = (lker p :&: lker q)%VS.
Proof.
apply/vspaceP => x.
rewrite memv_cap !memv_ker.
rewrite add_lfunE.
case Hpx: (p x == 0).
Admitted.

End b.

End Problem2.
```

Mathcomp の定理

```
(* ベクトル空間について *)
Lemma lkerE f U : (U <= lker f)%VS = (f @: U == 0)%VS.
Lemma subvv U : (U <= U)%VS.
Lemma subv0 U : (U <= 0)%VS = (U == 0%VS).
Lemma addv0 : right_id 0%VS addV.
Lemma capfv : left_id fullv capV.
Lemma subvf U : (U <= fullv)%VS.
Lemma memvf v : v \in fullv.
Lemma memvN U v : (- v \in U) = (v \in U).
Lemma memv_add u v U V : u \in U -> v \in V -> u + v \in (U + V)%VS.
Lemma memv_cap w U V : (w \in U :&: V)%VS = (w \in U) && (w \in V).
Lemma memv_img f v U : v \in U -> f v \in (f @: U)%VS.
Lemma memv_ker f v : (v \in lker f) = (f v == 0).
Lemma limg_ker_dim f U : (\dim (U :&: lker f) + \dim (f @: U) = \dim U)%N.
Lemma dimv_disjoint_sum U V :
  (U :&: V = 0)%VS -> \dim (U + V) = (\dim U + \dim V)%N.
Lemma dimv_eq0 U :  (\dim U == 0%N) = (U == 0%VS).
Lemma eqEdim U V : (U == V) = (U <= V)%VS && (\dim V <= \dim U).
Lemma eqEsubv U V : (U == V) = (U <= V <= U)%VS.
Lemma vspaceP U V : U =i V <-> U = V.

(* 環と体について *)
Lemma addr0 : right_id 0 +%R.
Lemma addrA : associative +%R.
Lemma addrC : commutative +%R.
Lemma subr_eq x y z : (x - z == y) = (x == y + z).
Lemma mulr2n x : x *+ 2 = x + x.
Lemma scaler_nat n v : n%:R *: v = v *+ n.
Lemma scaler_eq0 a v : (a *: v == 0) = (a == 0) || (v == 0).
Lemma linear0 (f : {linear U -> V | s}) : f 0 = 0.
Lemma linearD (f : {linear U -> V | s}) : {morph f : x y / x + y}.
Lemma Num.Theory.pnatr_eq0 n : (n%:R == 0 :> R) = (n == 0)%N.
```