EE951 Linear Algebra

**Inner products**

When we use the term "vector" we often refer to an array of numbers, and when we say "vector space" we refer to a set of such arrays.

a vector space is a **set** equipped with two operations, called vector addition and scalar multiplication, which satisfy a number of axioms.

the elements of the vector space are called vectors. In that abstract definition, a vector space has an associated field, which in most cases is the set of real numbers **ℝ** or the set of complex **ℂ** numbers.

**Definition:**

Let Ѕ be a vector space over ℂ. An inner product on Ѕ is a function **Ѕ x Ѕ → ℂ** that associates to each ordered pair of vectors **r,s ϵЅ** a complex number, denoted by **⟨r , s⟩** which has the following properties.

1. Positivity: **⟨r , r⟩ ≥ 0, ∀rϵS** where **⟨r , r⟩ ≥ 0** means that **⟨r , r⟩** is real (i.e., its complex part is

zero) and positive.

1. Definiteness: **⟨r , r⟩ = 0**, if and only if r = 0
2. Additivity in first argument: **⟨r + s, t⟩ = ⟨r , t⟩ + ⟨s , t⟩ , ∀ r,s,tϵS**
3. Homogeneity in first argument: **⟨αr, s⟩ = α⟨r , s⟩ , ∀ r,s ϵS, ∀ αϵ ℂ**
4. Conjugate symmetry: **⟨r , s⟩ = ⟨s , r⟩**

where **⟨s , r⟩** denotes the complex conjugate of **⟨r , s⟩**

**Example:**

**⟨r , s⟩ = sT. r = ∑rksk**  for all k=1…K

**Orthogonal vectors**

When the inner product between two vectors is equal to zero, that is, **⟨r , s⟩ = 0** then the two vectors are said to be orthogonal.

**Vector Norm**

**Definition** Let **Ѕ** be a vector space and **⟨. , .⟩** an inner product on **s** Then, the function **||s||** defined by **||s||** = √**⟨s,s⟩ , ∀ sϵЅ,** called an induced norm on **Ѕ.**

**Example:**

**⟨r , s⟩ = sT. r = ∑rksk**  for all k=1…K

let r = { {1},

{-1},

{2}

}

let s = { {2},

{0},

{1}

}

sT  = { {2}, {0}, {1}}

sT.r = { {2}, {0}, {1}} . { {1},

{-1},

{2}}

sT.r = {{2}.{1} + {0}.{-1} + {1}.{2}}

sT.r = {2 + 0 + 2} = {4}

**⟨r , s⟩ = {4}**

**C++ Code Example:**

#include <vector>

#include <iostream>

#include <cmath>

class Vector : public std::vector<std::vector<std::int32\_t>> {

public:

using vector::vector;

virtual ~Vector() = default;

Vector T();

std::int32\_t operator\* (const Vector& rhs) const;

Vector operator+ (const Vector& rhs) const;

Vector operator\* (const std::int32\_t& scalar);

friend Vector operator\* (const std::int32\_t& scalar, const Vector& v1);

long double N(std::uint32\_t order);

void print() const;

};

class VectorSpace {

public:

VectorSpace(Vector aa, Vector bb) {

m\_r = aa;

m\_s = bb;

}

VectorSpace() = default;

~VectorSpace() = default;

Vector r() const {

return(m\_r);

}

Vector s() const {

return(m\_s);

}

void r(Vector v) {

m\_r = v;

}

void s(Vector v) {

m\_s = v;

}

private:

Vector m\_r;

Vector m\_s;

};

#include "vectors.hpp"

**/// This computes the Norm of a Vector for a given order.**

long double Vector::N(std::uint32\_t order) {

if(this->empty()) {

std::cout << " Norm of Empty Vector" << std::endl;

return(0);

}

std::uint32\_t norm = 0;

auto rows = this->size();

for(auto row = 0; row < rows; ++row) {

for(auto ent: this->at(row)) {

norm += std::pow(ent, 2);

}

}

//std::cout << " The value of |v| is " << norm << std::endl;

if(norm) {

return(std::pow(norm, 1.0/order));

}

return(norm);

}

**/// This computes the Transpose of a Vector.**

Vector Vector::T() {

Vector \_mm;

if(this->empty()) {

/// @brief Transpose on Empty Matrix is not possible

return(\_mm);

}

std::vector<std::int32\_t> \_tt;

const auto& \_cols = this->at(0).size();

const auto& \_rows = this->size();

for(auto col = 0; col < \_cols; ++col) {

for(auto row = 0; row < \_rows; ++row) {

\_tt.push\_back(this->at(row).at(col));

}

\_mm.push\_back(\_tt);

\_tt.clear();

}

return(\_mm);

}

**/// This computes the Dot Product of two Vectors.**

std::int32\_t Vector::operator\* (const Vector& rhs) const {

auto lhs\_cols = this->at(0).size();

auto lhs\_rows = this->size();

auto rhs\_cols = rhs.at(0).size();

auto rhs\_rows = rhs.size();

if(this->empty() || rhs.empty() || (lhs\_rows != rhs\_cols && lhs\_cols != rhs\_rows)) {

/// @brief Matrix Multiplication not possible

std::cout << " Matrix multiplication is not possible" << std::endl;

return(-1);

}

std::int32\_t product = 0;

for(auto lhs\_row = 0; lhs\_row < lhs\_rows; ++lhs\_row) {

for(auto lhs\_col = 0; lhs\_col < lhs\_cols; ++lhs\_col) {

product += this->at(lhs\_row).at(lhs\_col) \* rhs.at(lhs\_col).at(lhs\_row);

}

}

return(product);

}

**/// This computes the addition of two Vectors.**

Vector Vector::operator+ (const Vector& rhs) const {

auto lhs\_cols = this->at(0).size();

auto lhs\_rows = this->size();

auto rhs\_cols = rhs.at(0).size();

auto rhs\_rows = rhs.size();

Vector v;

//std::cout << " lhs\_cols: " << lhs\_cols << " lhs\_rows: " << lhs\_rows << " rhs\_cols: " << rhs\_cols << " rhs\_rows: " << rhs\_rows << std::endl;

if(rhs.empty() || (lhs\_cols != rhs\_cols) && (lhs\_rows != rhs\_rows)) {

/// @brief Addition not possible

std::cout << " Vector Addition is not possible" << std::endl;

return(v);

}

std::vector<std::int32\_t> \_tt;

for(auto lhs\_row = 0; lhs\_row < lhs\_rows; ++lhs\_row) {

for(auto lhs\_col = 0; lhs\_col < lhs\_cols; ++lhs\_col) {

\_tt.push\_back(this->at(lhs\_row).at(lhs\_col) + rhs.at(lhs\_row).at(lhs\_col));

}

v.push\_back(\_tt);

\_tt.clear();

}

return(v);

}

**/// This computes the scalar product of a vector. Scalar number at right side of the vector.**

Vector Vector::operator\* (const std::int32\_t& scalar) {

return(scalar \* \*this);

}

**/// This computes the scalar product of a vector. Scalar will be at the left side of a vector.**

Vector operator\* (const std::int32\_t& scalar, const Vector& v1) {

auto lhs\_cols = v1.at(0).size();

auto lhs\_rows = v1.size();

Vector v;

if(v1.empty()) {

/// @brief scalar operation not possible on empty vector

return(v);

}

std::vector<std::int32\_t> \_tt;

for(auto lhs\_row = 0; lhs\_row < lhs\_rows; ++lhs\_row) {

for(auto lhs\_col = 0; lhs\_col < lhs\_cols; ++lhs\_col) {

\_tt.push\_back(v1.at(lhs\_row).at(lhs\_col) \* scalar);

}

v.push\_back(\_tt);

\_tt.clear();

}

//v.push\_back(\_tt);

return(v);

}

void Vector::print() const {

const auto& \_rows = this->size();

if(this->empty()) {

/// @brief Empty Vector

return;

}

std::cout << "{\n";

for(auto row = 0; row < \_rows; ++row) {

std::cout << "{";

const auto cols = this->at(row).size();

for(auto col = 0; col < cols; ++col) {

if(col < cols -1)

std::cout << " " << this->at(row).at(col) << " ";

else

std::cout << " " << this->at(row).at(col);

}

std::cout << "}\n";

}

std::cout << "}\n";

}

**⟨⟩∑≤||ϵЅ v**

**ǁR₵ℂℝ→**

**∀**

**∀ ≥**

**α**