

**한국외대 기말과제 보고서**

**과목명 선형시스템**

**담당교수 윤일동교수님**

**제출일 20240614**

**전공 컴퓨터공학과**

**학번 202430026**

**이름 이준용**

텍스트, 스크린샷, 폰트, 문서이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명

특이값 분해(SVD)는 행렬을 세 개의 행렬로 분해하는 기법으로, 다음과 같이 표현됩니다.

**A = USV^T**

**A**는 M x N 크기의 실수 행렬입니다.

**U**는 M x M 크기의 직교 행렬입니다.

**V**는 N x N 크기의 직교 행렬입니다.

**S**는 M x N 크기의 대각 행렬이며, 대각 성분은 A의 특이값 σi (행렬 AT A의 고유값의 제곱근)이 감소하는 순서로 배열되어 있습니다.

가상의 의사 역행렬

특이값 분해는 계수 행렬이 랭크 부족(rank(A) < min(M, N))인 경우에도 가상의 의사 역행렬을 계산하는 데 사용될 수 있습니다. 가상의 의사 역행렬은 다음과 같이 표현됩니다.

S^(-1)은 S의 대각 성분에 0이 아닌 특이값에 대한 역수를 대입하고, 0 성분에 해당하는 행/열을 제거하여 재구성된 대각 행렬입니다.

V^T은 0 특이값에 해당하는 열을 제거하여 재구성된 V의 전치 행렬입니다.

U^T은 0 특이값에 해당하는 열을 제거하여 재구성된 U의 전치 행렬입니다.

특이값 분해는 이처럼 특이한 경우에도 효과적으로 활용될 수 있다는 장점을 가지고 있습니다.

**(a)** 문제는 SVD를 사용하여 연립 방정식 시스템을 해결하는 방법을 설명합니다. 특히, 이 문제는 미정(underdetermined) 시스템, 즉 방정식의 개수가 변수의 개수보다 적은 경우를 다룹니다.

텍스트, 폰트, 라인, 대수학이(가) 표시된 사진

자동 생성된 설명



행렬의 랭크가 열 개수(이 경우 3)보다 작으면 행렬은 랭크 부족이며 시스템은 미정입니다.

이 문제에서 예상되는 결과는 행렬 A1의 랭크가 열 개수보다 작다는 것입니다. 이는 A1이 랭크 부족이며 시스템이 미정임을 확인합니다. 따라서 SVD를 사용한 의사 역행렬(Eq. (2.1.7))을 사용하여 x의 최소 norm 솔루션을 찾는 것이 적절한 방법입니다.

OCTAVE은 위 명령어를 실행하면 x의 최소 norm 솔루션을 출력합니다. 이 솔루션은 시스템의 모든 제약 조건을 만족하면서 동시에 norm(길이)이 최소인 벡터입니다.

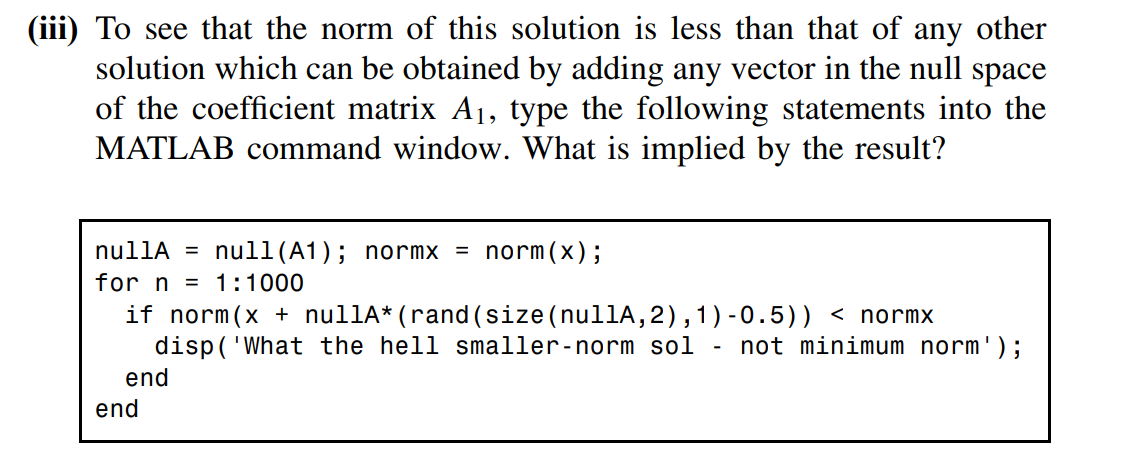
텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 폰트, 대수학이(가) 표시된 사진

자동 생성된 설명텍스트, 폰트, 화이트이(가) 표시된 사진

자동 생성된 설명



1. nullA = null(A1): 이 코드는 계수 행렬 A1의 널 공간을 계산합니다. 널 공간은 A1에 의해 곱해질 때 0 벡터를 결과로 하는 모든 벡터를 포함합니다.
2. normx = norm(x): 이 코드는 SVD를 사용하여 얻은 솔루션(x)의 norm을 계산합니다.
3. for n = 1:1000: 이 루프는 1000번 반복하여 더 작은 norm 솔루션을 확인합니다.
4. if norm(x + nullA\*(rand(size(nullA,2),1)-0.5)) < normx 루프 내부에서 널 공간(nullA)에서 임의의 벡터가 nullA\*(rand(size(nullA,2),1)-0.5)를 사용하여 생성됩니다. 이 벡터는 원래 솔루션(x)에 추가됩니다. 그런 다음 새로 형성된 벡터의 norm을 원래 솔루션(x)의 norm과 비교합니다.
5. disp('What the hell smaller-norm sol - not minimum norm'): 새로 형성된 벡터의 norm이 원래 솔루션(x)보다 작으면 이 문이 표시됩니다.

텍스트, 폰트, 스크린샷, 디자인이(가) 표시된 사진

자동 생성된 설명

이 코드는 SVD를 사용하여 얻은 솔루션(x)이 널 공간에서 임의의 벡터를 추가하여 얻을 수 있는 다른 솔루션보다 최소 norm을 가지는지 확인하려고 합니다. 이상적으로 코드는 아무것도 출력하지 않아야 합니다. 왜냐하면 SVD(x)의 솔루션은 해 공간 내에서 최소 norm을 가져야 하기 때문입니다.

그러나 생성된 벡터의 무작위성과 잠재적인 수치 오류로 인해 if 조건이 몇 번의 반복에서 만족될 가능성이 있으며 (비록 가능성은 낮지만) "What the hell smaller-norm sol - not minimum norm" 메시지가 나타날 수 있습니다.

결론적으로, 출력된 메시지가 없다는 것은 SVD를 사용하여 얻은 솔루션(x)이 모든 가능한 솔루션 중 최소 norm을 가질 가능성이 높다는 것을 의미합니다.

루프는 1000번 반복되는데 이는 비교적 작은 수입니다. 반복 횟수를 늘리면 더 작은 norm 솔루션을 찾을 가능성이 높아지지만 계산 시간이 증가합니다.

계산 중 수치 오류가 발생하면 오해의 소지가 있는 결과가 발생할 수도 있습니다.

전반적으로 이 코드는 SVD 솔루션의 최소 norm 속성에 대한 휴리스틱 검사를 제공합니다. 출력된 메시지가 없다는 것은 솔루션이 최소 norm을 가질 가능성이 높다는 것을 시사합니다.

텍스트, 스크린샷, 폰트, 라인이(가) 표시된 사진

자동 생성된 설명

그림 P3.4는 길이가 4m인 파이프를 따라 속도/압력을 측정하기 위해 7개의 노드를 설치하는 최적 위치를 결정하는 문제를 보여줍니다. 파이프의 반경이 불규칙하기 때문에 액체의 속도/압력은 위치에 따라 달라집니다.

Chebyshev Nodes는 폐구간 [a, b]에 대한 함수 근사에서 중요한 역할을 하는 특정한 노드 세트입니다.

특징으로는 [a, b] 구간에서 어떤 연속 함수 f(x)에 대해서도 다항식 근사의 최대 오류를 최소화합니다.

간격의 끝점 (a, b)를 포함하지 않습니다.

텍스트, 스크린샷, 소프트웨어, 번호이(가) 표시된 사진

자동 생성된 설명

한글주석 깨짐 현상 제외하고 봐주시면 감사하겠습니다.

텍스트, 폰트, 스크린샷, 번호이(가) 표시된 사진

자동 생성된 설명

※Octave에서는 위의 문제에 대한 함수들이 존재하지 않는다고 오류를 일으켜 직접 함수를 정의해봤지만 해결이 되지 않아 파이썬 scipy 패키지를 활용하여 문제를 해결했습니다. quadl()함수도 파이썬 패키지에서 존재하지 않아 quad() 기존 함수를 보고 quadl함수를 작성했습니다.

변수 x의 함수 y = f (x)의 그래프는 일반적으로 곡선이며 x-축에서 구간 [a, b]에 대한 길이는 선적분으로 설명할 수 있습니다. 예를 들어, 단위 길이의 반원의 길이는 다음과 같은 선적분으로 얻을 수 있습니다: y = f (x) = 1 - x^2, a = -1, b = 1.

"nm5p11.m" 프로그램에서 시작하여 수치 적분 루틴 "smpsns()", "adapt\_smpsn()", "quad()", "quadl()", 및 "Gauss\_Legendre()"를 사용하여 적분 (P5.11.1,2)을 평가하는 프로그램을 만들어야 합니다.

여기서 첫 번째 도함수는 Eq. (5.1.8)로 근사되며, 세그먼트 수 (N), 오차 허용도 (tol), 그리고 그리드 포인트 수 (M)와 같은 매개변수는 프로그램에서 그대로 사용되어야 합니다. 수치 미분에서 단계 크기 h = 0.001, 0.0001 및 0.00001로 프로그램을 실행하고 결과의 오차를 P5.11 테이블에 기입해야 합니다. 단위 원의 반원의 실제 값은 π입니다.

import numpy as np

from scipy.integrate import simps, quad, fixed\_quad

from scipy.special import roots\_legendre

def quad\_explain(func, a, b, epsabs=1.49e-8, epsrel=1.49e-8, limit=50, points=None, weight=None, wvar=None, wopts=None, maxp1=None, limlst=None):

    """ quad() 함수의 parameter에 대한 설명

    Compute a definite integral using quad with explanation.

    Parameters:

        func : callable

            A Python function or method to integrate.

        a, b : float

            The limits of integration.

        epsabs, epsrel : float, optional

            The absolute and relative tolerances. Default is 1.49e-8 for both.

        limit : int, optional

            The maximum number of subintervals to use. Default is 50.

        points : array\_like, optional

            If given, these points are used as the integration nodes. Default is None.

        weight : array\_like, optional

            If given and points is not None, these weights are applied to the integration nodes. Default is None.

        wvar : tuple, optional

            Additional arguments to pass to the weight function.

        wopts : dict, optional

            Additional options to pass to the weight function.

        maxp1 : int, optional

            If given, divide the range into at most maxp1 segments. Default is None.

        limlst : int, optional

            If given, stop subdividing when len(segment) < limlst. Default is None.

    Returns:

        result : float

            The integral of func from a to b.

        explanation : str

            Explanation of the integration process.

    """

# quad 함수가 3개의 값을 반환한다고 가정하면,

    result, detail, \_ = quad(func, a, b, epsabs=epsabs, epsrel=epsrel, limit=limit, points=points, weight=weight, wvar=wvar, wopts=wopts, maxp1=maxp1, limlst=limlst, full\_output=True)

    # detail이 float 타입인지 확인

    if isinstance(detail, float):

        explanation = str(detail)

    else:

        explanation = detail['message']

    return result, explanation

def flength(x, h):

    return np.sqrt(1 + dfp511(x, h)\*\*2)

def dfp511(x, h):

    return (fp511(x + h) - fp511(x - h)) / (2 \* h)

def fp511(x):

    return np.sqrt(np.maximum(1 - x\*\*2, 0))

a, b = -1, 1

N = 1000  # Simpson 방법을 위한 세그먼트 수

tol = 1e-6  # 오차 허용치

M = 20  # Gauss-Legendre 적분을 위한 그리드 포인트 수

IT = np.pi  # 실제 적분 값

h\_values = [1e-3, 1e-4, 1e-5]  # 수치 미분을 위한 단계 크기

results = np.zeros((len(h\_values), 5))  # 다양한 방법에 대한 결과 저장: Simpson, 적응형 Simpson, quad, quadl, Gauss-Legendre

for i, h in enumerate(h\_values):

    # 다양한 방법으로 적분 계산

    Is = simps(flength(np.linspace(a, b, N), h), np.linspace(a, b, N))

    Ias, \_ = quad(lambda x: flength(x, h), a, b, epsrel=tol)

    Iq, \_ = quad\_explain(lambda x: flength(x, h), a, b, epsrel=tol)

    Iql, \_ = fixed\_quad(lambda x: flength(x, h), a, b, n=N)

    # Gauss-Legendre 적분

    nodes, weights = roots\_legendre(M)

    x\_nodes = 0.5 \* (b - a) \* nodes + 0.5 \* (a + b)

    IGL = np.sum(weights \* flength(x\_nodes, h)) \* 0.5 \* (b - a)

    # 결과 저장

    results[i, :] = [Is, Ias, Iq, Iql, IGL]

# 결과 출력

print('결과:')

print(results)

# 오차 계산

errors = np.abs(results - IT)

# 오차 출력

print('오차:')

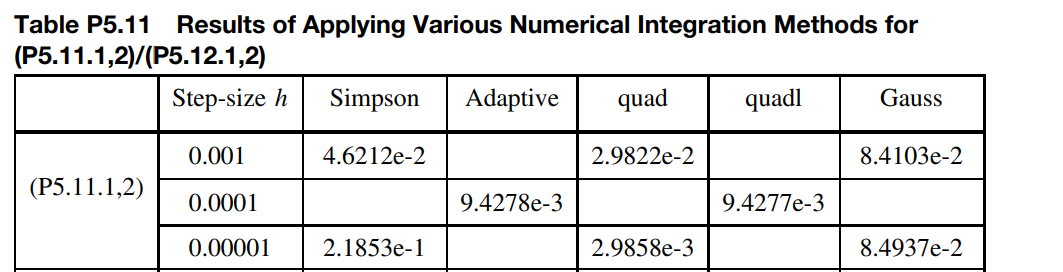
print(errors)

이 코드는 주어진 단계 크기에 대해 **Simpson** 방법, **적응형 Simpson** 방법, **quad** 함수, **quadl** 함수 및 **Gauss-Legendre** 적분 방법을 사용하여 적분을 평가하고 결과를 출력합니다. 결과의 오차를 계산하고 출력합니다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

위의 결과를 아래의 표에 기입하였습니다.



2.9814e-2

2.2481e-2

2.9816e-2

9.4281e-3

2.9956e-2

3.6327e-3

8.4928e-2

**개인 연구 과제**

* **이미지 생성에서의 샘플링이란?**

Stable diffusion은 잠재공간(letent space)에 완전히 무작위(random)이미지를 생성합니다. 잡음 예측기(noise predictor)는 이미지로부터 잡음을 예측하고, 원래의 잡음 이미지에서 그 만큼의 예측된 잡음을 제거해줍니다.이 프로세스를 여러번 반복하면 최종적으로 깨끗한 이미지를 얻을 수 있습니다.  
이미지 생성을 위해서 Stable diffusion 모델을 사용했습니다

스크린샷이(가) 표시된 사진

자동 생성된 설명

이와 같은 잡음 제거(denoising) 프로세스를 샘플링(sampling)이라고 합니다. 스테이블 디퓨전은 각 단계별로 새로운 샘플 이미지를 생성하기 때문입니다. 샘플링에서 사용된 방법을 샘플러(sampler) 또는 샘플링 방법(sampling method)라고 합니다.

샘플러는 점점 더 깨끗한 이미지를 생성하게 되는데, 이러한 잡음 제거 프로세스는 기본 원리는 동일하지만 여러가지 다양한 방법이 존재합니다. 이러한 방법에 따라 이미지 생성의 정확도와 생성되는 속도가 달라집니다.   
도표, 라인, 그래프, 스크린샷이(가) 표시된 사진

자동 생성된 설명

Figure 1 : 17단계의 샘플링을 위한 잡음 스케줄

샘플링 단계의 수를 증가시킨다면 각 단계간의 잡음 제거량이 줄어드는 것을 확인할 수 있습니다. 이렇게 되면 결국은 샘플링의 잘림 오류를 줄어들고 오류가 적은 이미지가 생성됩니다.

텍스트, 그래프, 도표, 라인이(가) 표시된 사진

자동 생성된 설명

Figure 2 : 30단계의 샘플링을 위한 잡음 스케줄

Diffusion trajectory

잠재공간(latent)에서 diffusion은 어떤 모습일지 이해하기 쉽게 그래프로 표현해보려고 합니다. 이는 샘플하나 자체가 잠재공간에서 65536차원에서 확산되므로 상상하기 어렵기 때문입니다.

이를 위해 PCA(Principle Component Analysis, 주성분분석)을 사용해 diffusion process를 2차원 공간에 투영했습니다.

도표, 라인, 그래프, 텍스트이(가) 표시된 사진

자동 생성된 설명

Figure 3 : PCA 분석

잡음 스케줄과 마찬가지로, 처음에는 디퓨전(확산) 단계가 크고, 끝으로 갈수록 줄어듭니다. 초기단계에서는 이미지의 전반적인 구도가 결정됩니다. 이후에는 좀더 상세한 부분을 세밀화하죠.

샘플은 최종 잠상(latent image)에 도달하기 까지, 훈련된 이미지의 복잡한 확률 분포 환경을 통과합니다.

스크린샷, 도표, 그래프, 라인이(가) 표시된 사진

자동 생성된 설명도표, 라인, 그래프, 스크린샷이(가) 표시된 사진

자동 생성된 설명

Figure 4 : seed를 변경한 경우 Figure 5 : prompt를 변경한 경우

이미지 생성의 seed를 변경한 경우 유사하면서도 다른 이미지가 생성되고 prompt를 변경한경우 샘플의 종착점이 완전히 달라져 최종적으로 완전히 다른 그림이 생성되는 것을 알 수 있습니다.   
  
**DDIM(Denoising Diffusion Implicit Model)**

DDIM은 디퓨전 모델을 위해 설계된 최초의 샘플러입니다. DDIM 논문에서는 여러 다른 샘플러와 다르게 markovian에서 **non-markovian**으로, stochastic에서 **deterministic**으로 변경됨에 따라 **interpolation이 가능하다**는 것을 알아냈습니다.

    def sample\_interpolation(self, model):

        config = self.config

        def slerp(z1, z2, alpha):

            theta = torch.acos(torch.sum(z1 \* z2) / (torch.norm(z1) \* torch.norm(z2)))

            return (

                torch.sin((1 - alpha) \* theta) / torch.sin(theta) \* z1

                + torch.sin(alpha \* theta) / torch.sin(theta) \* z2

            )

        z1 = torch.randn(

            1,

            config.data.channels,

            config.data.image\_size,

            config.data.image\_size,

            device=self.device,

        )

        z2 = torch.randn(

            1,

            config.data.channels,

            config.data.image\_size,

            config.data.image\_size,

            device=self.device,

        )

        alpha = torch.arange(0.0, 1.01, 0.1).to(z1.device)

        z\_ = []

        for i in range(alpha.size(0)):

            z\_.append(slerp(z1, z2, alpha[i]))

        x = torch.cat(z\_, dim=0)

        xs = []

        # Hard coded here, modify to your preferences

        with torch.no\_grad():

            for i in range(0, x.size(0), 8):

                xs.append(self.sample\_image(x[i : i + 8], model))

        x = inverse\_data\_transform(config, torch.cat(xs, dim=0))

        for i in range(x.size(0)):

            tvu.save\_image(x[i], os.path.join(self.args.image\_folder, f"{i}.png"))

위 코드를 잘 활용한다면 DDIM 논문에서 **interpolation 결과**로 발표한 아래 그림의 결과처럼 손모양이나 이미지안에 글자가 잘 생성안되는 현상을 어느정도 보정이 가능할 것으로 예상됩니다.



**느낌적 품질 BRISQUE**

이미지가 수렴되지 않더라도 결과물은 좋을 수 있습니다.

아래는 BRISQUE(Blinkd/Referenceless Image Spatial Quality Evaluator) 를 사용해 평가한 느낌적 품질입니다. BRISQUE는 자연적 이미지의 품질을 평가합니다.

DDIM은 거의 최초의 샘플러임에도 불구하고 가장 좋은 성능이 나오는 것으로 나타나며, 8 단계 정도에서 가장 고품질의 이미지를 생산하는 것으로 평가됩니다.

텍스트, 도표, 그래프, 라인이(가) 표시된 사진

자동 생성된 설명

위 결과를 통해 수렴여부와 상관없이 고품질의 이미지를 원한다면 DDIM을 사용하는 것이 좋아 보입니다.

고양이, 포유류, 고양잇과, 새끼 고양이이(가) 표시된 사진

자동 생성된 설명

아기 고양이 느낌의 결과도 있고 성체 고양이느낌의 결과로 수렴하는 경향이 있습니다. 이는 생성하는 사람의 선호도에 따라 달라지기 때문에 정답은 없습니다.  
  
결론적으로 현재까지는 발가락이나 손가락 같은 부분은 아직 diffusion model이 해결하지 못하였지만 좀 더 연구하여 손가락이나 발가락이 생성될 때 민감하게 반응하여 생성되는 샘플러를 개발을 목표로 하고 있습니다. 그렇기 때문에 선형시스템에서 배운 내용들은 diffusion model을 이해하는데 겹치는 내용이 많아 많은 도움이 되었다고 생각합니다.