CHAPTER 1

ASSIGNMENT 1

1.1 Question 1

(a) Write down the definition of **norm** for a vector space.

Solution 1

Let $\mathbf{X}$ be a real vector space. A nonnegative-valued function $\|\cdot\| : \mathbf{X} \to \mathbb{R}$ is called a norm if $\forall \mathbf{x}, \mathbf{y} \in \mathbf{X}, \lambda \in \mathbb{R}$

- $\|\lambda \mathbf{x}\| = |\lambda| \|\mathbf{x}\|$ (absolutely homogeneous);
- $\|\mathbf{x} + \mathbf{y}\| \leq \|\mathbf{x}\| + \|\mathbf{y}\|$ (triangle inequality);
- $\|\mathbf{x}\| = 0 \Rightarrow \mathbf{x} = \vec{\mathbf{0}}$ (positive definite).

Moreover, the vector norm $\|\cdot\|_p$ for $p = 1, 2, \ldots$ is defined as

$$\|\mathbf{x}\|_p = \left(\sum_{1 \leq i \leq n} |x_i|^p \right)^{1/p}.$$

And the pair $(\mathbf{X}, \|\cdot\|)$ is called a normed space.

(b) Given $\mathbf{x} \in \mathbb{R}^n$, show that followings are norm on $\mathbb{R}^n$.

i. $\|\mathbf{x}\|_\infty = \max_{1 \leq i \leq n} |x_i|$;

Solution 2

- $\|\lambda \mathbf{x}\|_\infty = \max_{1 \leq i \leq n} |\lambda x_i| = |\lambda| \max_{1 \leq i \leq n} |x_i| = |\lambda| \|\mathbf{x}\|_\infty$
- $\|\mathbf{x} + \mathbf{y}\|_\infty = \max_{1 \leq i \leq n} |x_i + y_i| \leq \max_{1 \leq i \leq n} |x_i| + \max_{1 \leq i \leq n} |y_i| = \|\mathbf{x}\|_\infty + \|\mathbf{y}\|_\infty$
- $\|\mathbf{x}\|_\infty = \max_{1 \leq i \leq n} |x_i| = 0 \Rightarrow |x_i| = 0$ where $1 \leq i \leq n \Rightarrow \mathbf{x} = \vec{\mathbf{0}}$

ii. $\|\mathbf{x}\|_1 = \sum_{1 \leq i \leq n} |x_i|$;

Solution 3

- $\|\lambda \mathbf{x}\|_1 = \sum_{1 \leq i \leq n} |\lambda x_i| = |\lambda| \sum_{1 \leq i \leq n} |x_i| = |\lambda| \|\mathbf{x}\|_1$
- $\|\mathbf{x} + \mathbf{y}\|_1 = \sum_{1 \leq i \leq n} |x_i + y_i| \leq \sum_{1 \leq i \leq n} |x_i| + \sum_{1 \leq i \leq n} |y_i| = \|\mathbf{x}\|_1 + \|\mathbf{y}\|_1$

- $\|\mathbf{x}\|_1 = \sum_{1\le i\le n} |x_i| = 0 \Rightarrow |x_i| = 0$ where $1 \le i \le n \Rightarrow \mathbf{x} = \vec{\mathbf{0}}$

iii. $\|\mathbf{x}\|_2 = \left(\sum_{1\le i\le n} |x_i|^2\right)^{1/2}$.

Solution 4

- $\|\lambda\mathbf{x}\|_2 = \left(\sum_{1\le i\le n} |\lambda x_i|^2\right)^{1/2} = |\lambda|\left(\sum_{1\le i\le n} |x_i|^2\right)^{1/2} = |\lambda|\|\mathbf{x}\|_2$
- $\|\mathbf{x}+\mathbf{y}\|_2 = \left(\sum_{1\le i\le n} |x_i+y_i|^2\right)^{1/2} \le \left(\sum_{1\le i\le n} |x_i|^2\right)^{1/2} + \left(\sum_{1\le i\le n} |y_i|^2\right)^{1/2} = \|\mathbf{x}\|_2 + \|\mathbf{y}\|_2$
- $\|\mathbf{x}\|_2 = \left(\sum_{1\le i\le n} |x_i|^2\right)^{1/2} = 0 \Rightarrow \sum_{1\le i\le n} |x_i|^2 \Rightarrow \mathbf{x} = \vec{\mathbf{0}}$

Plot the regions for $\|\mathbf{x}\|_p \le 1$ in $\mathbb{R}^2$ for $p = \infty, 1, 2$ respectively.

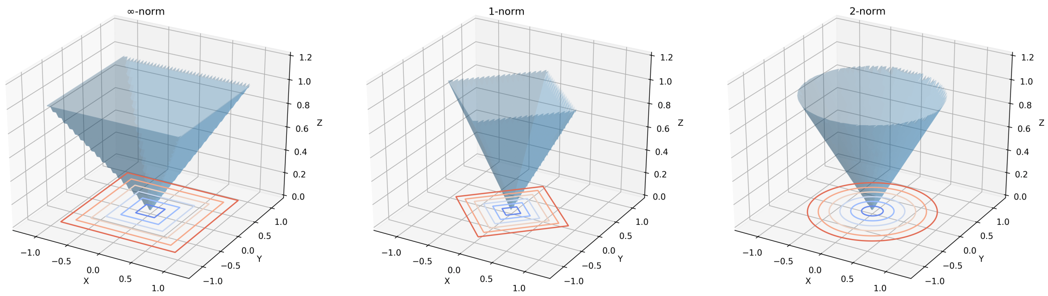

Figure 1.1: Please refer to Appendix 1.6

(c) Given the vector norm $\|\cdot\|_p$ for $\mathbb{R}^n$, the induced norm for matrices $\mathbf{A} \in \mathbb{R}^{n\times n}$ is defined as

$$\|\mathbf{A}\|_p = \max_{\mathbf{x}\neq 0} \frac{\|\mathbf{A}\mathbf{x}\|_p}{\|\mathbf{x}\|_p} = \max_{\|\mathbf{x}\|_p=1} \|\mathbf{A}\mathbf{x}\|_p.$$

Show that

i. $\|\mathbf{A}\|_\infty = \max_{1\le i\le n} \sum_{j=1}^n |a_{ij}|$;

Solution 5

$$\begin{aligned}
\|\mathbf{A}\|_\infty &= \max_{\|\mathbf{x}\|_\infty=1} \|\mathbf{A}\mathbf{x}\|_\infty \\
&= \max_{\max_{1\le j\le n}|x_j|=1} \max_{1\le i\le n} \sum_{j=1}^n |a_{ij}x_j| \\
&\le \max_{1\le i\le n} \sum_{j=1}^n |a_{ij}|
\end{aligned}$$

The equality holds when $x_j = \pm 1$ for $j = 1, 2, \dots, n$.

ii. $\|\mathbf{A}\|_1 = \max_{1\le j\le n} \sum_{i=1}^n |a_{ij}|$;

Solution 6

$$\begin{aligned}
\|\mathbf{A}\|_1 &= \max_{\|\mathbf{x}\|_1=1} \|\mathbf{A}\mathbf{x}\|_1 \\
&= \max_{\sum_{j=1}^n |x_j|=1} \sum_{i=1}^n \sum_{j=1}^n |a_{ij}x_j| \\
&\leq \max_{\sum_{j=1}^n |x_j|=1} \sum_{j=1}^n \sum_{i=1}^n |a_{ij}||x_j| = \max_{\sum_{j=1}^n |x_j|=1} \sum_{j=1}^n |x_j| \sum_{i=1}^n |a_{ij}| \\
&\leq \max_{\sum_{j=1}^n |x_j|=1} \sum_{j=1}^n |x_j| \max_{1\leq j\leq n} \sum_{i=1}^n |a_{ij}| = \max_{1\leq j\leq n} \sum_{i=1}^n |a_{ij}|
\end{aligned}$$

The equality holds when $j = \arg\max_{1\leq j\leq n} \sum_{i=1}^n |a_{ij}|$.

iii. $\|\mathbf{A}\|_2 = \sqrt{\lambda_{\max}(\mathbf{A}^T\mathbf{A})}$.

Solution 7

$\mathbf{A}$ can be decomposed as $\mathbf{U}\Sigma\mathbf{V}^T$ by singular value decomposition, and the singular values on the diagonal are decreasing from top left to bottom right.

$$\begin{aligned}
\|\mathbf{A}\|_2^2 &= \max_{\|\mathbf{x}\|_2=1} \|\mathbf{A}\mathbf{x}\|_2^2 \\
&= \max_{\|\mathbf{x}\|_2=1} (\mathbf{A}\mathbf{x})^T\mathbf{A}\mathbf{x} = \max_{\|\mathbf{x}\|_2=1} \mathbf{x}^T\mathbf{V}\Sigma^2\mathbf{V}^T\mathbf{x} \\
&= \max_{\|\mathbf{x}\|_2=1} \mathbf{y}^T\Sigma\mathbf{y} = \max_{\|\mathbf{x}\|_2=1} \sum_{i=1}^n \sigma_i y_i^2 \\
&\leq \max_{\|\mathbf{x}\|_2=1} \max_{1\leq i\leq n} \sigma_i \mathbf{y}^T\mathbf{y} = \max_{\|\mathbf{x}\|_2=1} \max_{1\leq i\leq n} \sigma_i \mathbf{x}^T\mathbf{x} \\
&= \max_{1\leq i\leq n} \sigma_i = \lambda_{\max}(\mathbf{A}^T\mathbf{A}).
\end{aligned}$$

The equality holds when $\mathbf{x} = \mathbf{V}\mathbf{e}_1$, where $\mathbf{e}_1 = [1, 0, \dots, 0]^T$.

1.2 Question 2

(a) Use the method of undetermined coefficients to design third order accurate approximation to $u'(\bar{x})$ by using the discrete points $\bar{x}+h$, $\bar{x}$, $\bar{x}-h$, $\bar{x}-2h$.

Solution 8

According to Appendix 1.6, we have

$$u'(\bar{x}) = \frac{1}{h}\left(\frac{u(\bar{x}-2h)}{6} - u(\bar{x}-h) + \frac{u(\bar{x})}{2} + \frac{u(\bar{x}+h)}{3}\right).$$

(b) Assuming $u(x)$ is smooth enough, compute the truncation error (leading term) for the finite difference formula above.

Solution 9

The leading term of the truncation error is of $\mathcal{O}(h^3)$, that is

$$\frac{h^3}{12}u^{(4)}(\bar{x}).$$

(c) What is the truncation error if $u(x) = 4x^4 + 12x^3 + 6x^2 + x/2 + \pi$.

Solution 10

$$\begin{aligned} u'(\bar{x}) &= \frac{1}{h}\left(\frac{u(\bar{x}-2h)}{6} - u(\bar{x}-h) + \frac{u(\bar{x})}{2} + \frac{u(\bar{x}+h)}{3}\right) \\ &= 16\bar{x}^3 + 36\bar{x}^2 + 12\bar{x} + \frac{1}{2} \end{aligned}$$

We have $u'(x) = 16x^3 + 36x^2 + 12x + 1/2$, that is, the truncation error of this given function is 0.

1.3 Question 3

(a) For the following $N \times N$ matrix, show that all eigenvalues are given by $\lambda_p = a + 2b \cos \frac{\pi p}{N+1}$ for $p = 1, 2, \ldots, N$.

$$\begin{bmatrix} a & b & & \\ b & \ddots & \ddots & \\ & \ddots & \ddots & b \\ & & b & a \end{bmatrix}_{N \times N}$$

Solution 11

Please refer to Section 1.4.

1.4 Question 4

(a) For the following $N \times N$ matrix, show that all eigenvalues are given by $\lambda_p = a + 2\sqrt{bc} \cos \frac{\pi p}{N+1}$ for $p = 1, 2, \ldots, N$ and $bc > 1$.

$$\mathbf{A} = \begin{bmatrix} a & c & & \\ b & \ddots & \ddots & \\ & \ddots & \ddots & c \\ & & b & a \end{bmatrix}_{N \times N}$$

Solution 12

The eigenvalues are $\lambda_p = a + 2\sqrt{bc} \cos \frac{\pi p}{N+1}$, and the corresponding eigenvector $\mathbf{x}^p$ has jth component

$$\mathbf{x}_j^p = \left(\sqrt{\frac{b}{c}}\right)^j \sin\left(\frac{p\pi j}{N+1}\right).$$

And $\mathbf{x}_0^p = \mathbf{x}_{N+1}^p = 0$, if we want to show that $\mathbf{A}\mathbf{x}^p = \lambda_p \mathbf{x}_p$, then we just need to show that $b\mathbf{x}_{j-1}^p + a\mathbf{x}_j^p + c\mathbf{x}_{j+1}^p = \lambda_p x_j^p$ for $j = 1, 2, \ldots, N$. We use the method of subtracting two sides to determine whether the two sides are equal by judging whether the result is zero. That is,

$$
\begin{aligned}
&b\mathbf{x}_{j-1}^p + a\mathbf{x}_j^p + c\mathbf{x}_{j+1}^p - \lambda_p x_j^p \\
=&b\mathbf{x}_{j-1}^p - 2\sqrt{bc}\cos\frac{\pi p}{N+1}\mathbf{x}_j^p + c\mathbf{x}_{j+1}^p \\
=&\left(\frac{b}{c}\right)^{(j-1)/2}\left(b\sin\left(\frac{\pi p(j-1)}{N+1}\right) + b\sin\left(\frac{\pi p(j+1)}{N+1}\right) - 2b\sin\left(\frac{\pi pj}{N+1}\right)\cos\left(\frac{\pi p}{N+1}\right)\right) \\
=&\left(\frac{b}{c}\right)^{(j-1)/2} b\left(\sin\left(\frac{\pi p(j-1)}{N+1}\right) + \sin\left(\frac{\pi p(j+1)}{N+1}\right) - 2\sin\left(\frac{\pi pj}{N+1}\right)\cos\left(\frac{\pi p}{N+1}\right)\right) \\
=&0.
\end{aligned}
$$

The last step relies on $\sin(x+y) = \sin(x)\cos(y) + \cos(x)\sin(y)$. Since $N \times N$ matrix has at most N eigenvalues, then $\lambda_p = a + 2\sqrt{bc}\cos\frac{\pi p}{N+1}$ is all the eigenvalues the matrix has.

1.5 Question 5

For the 2-point BVP with Neumann boundary conditions:

$$
\begin{cases} u''(x) = f(x), & x \in (0,1) \\ u'(0) = u'(1) = 0 \end{cases}
$$

(a) Set the grid points as $x_j = jh$ for $j = 0, 1, \dots, N$ and $h = 1/N$. Write down the central FD scheme for the main equation with using u_j to approximate $u(x_j)$.

Solution 13

According to the central finite difference scheme, for $j = 1, 2, \dots, N-1$ we have

$$
u''(x_j) = \frac{u(x_{j+1}) - 2u(x_j) + u(x_{j-1})}{h^2} = f(u_j)
$$

If we use $u'(\bar{x}) = (u(\bar{x}+h) - u(\bar{x}))/h$ or $u'(\bar{x}) = (u(\bar{x}) - u(\bar{x}) - h)/h$, we can approximate $u(x_0)$ by $u(x_1)$ and $u(x_N)$ by $u(x_{N-1})$. That is

$$
\frac{1}{h^2}\begin{bmatrix} -1 & 1 & & & \\ 1 & -2 & \ddots & & \\ & \ddots & \ddots & \ddots & \\ & & \ddots & -2 & 1 \\ & & & 1 & -1 \end{bmatrix}_{N-1\times N-1} \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ \vdots \\ u_{-1} \end{bmatrix} = \begin{bmatrix} f(x_1) \\ f(x_2) \\ \vdots \\ \vdots \\ f(x_{N-1}) \end{bmatrix}.
$$

(b) Add two "ghost" points as $x_{-1} = -h$ and $x_{N+1} = 1 + h$, and two more variables u_{-1} and u_{N+1}. Treat the boundary condition as $(u_1 - u_{-1})/2h = 0$, $(u_{N+1} - u_{N-1})/2h = 0$. Assemble all the equations as $\mathbf{AU} = \mathbf{F}$ where $\mathbf{U} = [u_0, u_1, \dots, u_N]^T$.

Solution 14

If we treat the boundary condition as the given way, we can approximate $u(x_{-1})$ by $u(x_1)$ and $u(x_{N+1})$ by $u(x_{N-1})$. Then all the equations can be assemble as $\mathbf{AU} = \mathbf{F}$ where

$\mathbf{U} = [u_0, u_1, \dots, u_N]^T$ and $\mathbf{F} = [f(x_0), f(x_1), \dots, f(x_N)]^T$. Also, we have

$$\mathbf{A} = \frac{1}{h^2}\begin{bmatrix} -2 & 2 & & & \\ 1 & \ddots & 1 & & \\ & \ddots & \ddots & \ddots & \\ & & 1 & \ddots & 1 \\ & & & 2 & -2 \end{bmatrix}_{N+1\times N+1}$$

(c) Show that $\mathbf{A}$ is singular, and find out when the system $\mathbf{AU} = \mathbf{F}$ has solutions.

Solution 15

We can first check the validity of the conclusion by MATLAB.

```
A = @(N) -2*eye(N+1) + diag([2, ones(1, N-1)], 1) + diag([ones(1,
    N-1), 2], -1);
for N = 1 : 99
    if det(A(N)) ~= 0
        disp(N);
    end
end
```

Then, we notice that the row sum of $\mathbf{A}$ is 0, which means rank $\mathbf{A} \neq N$, this leads to the conclusion that $\mathbf{A}$ is singular. As for when the system $\mathbf{AU} = \mathbf{F}$ has solutions, through the knowledge of linear algebra, we know that $\mathbf{F}$ must in the range space[a] of $\mathbf{A}$.

[a]In linear algebra, range space refers to the column space of a matrix, which is the set of all possible linear combinations of its column vectors.

(d) Find the kernel space of $\mathbf{A}$.

Solution 16

The kernel space[a] is also called the null space, which is defined as

$$\operatorname{null} \mathbf{A}_{n\times n} = \left\{ \mathbf{x} \in \mathbb{R}^n \mid \mathbf{Ax} = \vec{\mathbf{0}} \right\}.$$

Therefor, the kernel space of $\mathbf{A}$ is

$$\operatorname{span}\left\{ \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix}_{N+1\times 1} \right\}.$$

[a]The definition of kernel space is also appears in the computer field: System memory in Linux can be divided into two distinct regions: kernel space and user space. Kernel space is where the kernel (i.e., the core of the operating system) executes (i.e., runs) and provides its services.

(e) Find out all eigenvalues and eigenvectors for the matrix $\mathbf{A}$ given in (b).

todo: re-derive it later.

Solution 17

The non-zero eigenvalues of the matrix $\mathbf{A}_{N+1\times N+1}$ is

$$\lambda_p = -2 + 2\cos\frac{\pi p}{N},$$

and the corresponding eigenvector $\mathbf{x}^p$ has jth component

$$\mathbf{x}_j^p = \cos\left(\frac{p\pi(j-1)}{N}\right)$$

for $p = 0, 1, \ldots, N$. We can then check the correctness of the result by MATLAB.

```
A = @(N) -2*eye(N+1) + diag([2, ones(1, N-1)], 1) + diag([ones(1, N-1), 2], -1);
lambda = @(N, p) -2 + 2*cos(pi*p/N);
x = @(N, p) cos(pi*p*(0:N)/N)';
error = @(N) N^2 * eps;
for N = 1 : 99
    for p = 0 : N
        delta = A(N)*x(N, p) - lambda(N, p)*x(N, p);
        if norm(delta, 1) > error(N)
            disp(['N=', num2str(N), '; p=', num2str(p), '; error=', num2str(norm(delta, 1))]);
        end
    end
end

```

1.6 Appendix

hw1.py

```
def question_1():
    # packages
    import matplotlib.pyplot as plt
    from mpl_toolkits.mplot3d import axes3d
    from matplotlib import cm
    import numpy as np

    # parameters
    t = np.linspace(-1.2, 1.2, 512)
    X, Y = np.meshgrid(t, t)
    norms = {
        '$\infty$-norm': lambda x, y: max(abs(x), abs(y)),
        '1-norm': lambda x, y: abs(x)+abs(y),
        '2-norm': lambda x, y: np.sqrt(x**2+y**2),
    }
    zlim = 0, 1.2
    figsize = 9, 3

    # body
    length = len(norms)
    Z = np.zeros_like(X)
    fig = plt.figure(figsize=figsize)
    for ith, (name, func) in enumerate(norms.items()):
        for jth in range(Z.size):
            value = func(X.flat[jth], Y.flat[jth])
            Z.flat[jth] = value if value<=1 else np.nan
        ax = fig.add_subplot(1, length, ith+1, projection='3d')
        ax.plot_surface(X, Y, Z, alpha=0.3)
        ax.contour(X, Y, Z, zdir='z', offset=zlim[0], cmap=cm.coolwarm)
        ax.set_zlim(*zlim)
        ax.set_xlabel('X')
        ax.set_ylabel('Y')
        ax.set_zlabel('Z')
        ax.set_title(name)
    plt.show()


if __name__ == '__main__':
    question_1()
```

1_2.py

```
#!/usr/bin/python3
import numpy as np


def deriving_finite_difference_approximations(hs, order=1, errors=1):
    '''Deriving finite difference approximations
    using the method of undetermined coefficients.

    Example
        >>> # Du(x) = au(x) + bu(x-h) + cu(x-2h)
```

```
        >>> hs = [0, -1, -2]
        >>> order = 1
        >>> deriving_finite_difference_approximations(hs, order)
        ([1.5, -2.0, 0.5], [-0.3333333333333333])
        >>> # a, b, c = (1.5, -2.0, 0.5) / h^order
        >>> # Du(x)-u'(x) = -0.333 * h^2 * u^(3)(x)
        >>> # h^2: 2 = len(hs) + ith - order where errors[ith]=-0.333
    '''
    # the method of undetermined coefficients
    length = len(hs)
    _hs = np.ones(length)
    A = np.ndarray((length, length))
    A[0, :] = 1
    for ith in range(1, length):
        _hs *= hs
        A[ith, :] = _hs
    b = np.zeros(length)
    b[order] = np.math.factorial(order)
    coefficients = np.linalg.solve(A, b)
    # errors
    _errors = [None] * errors
    denominator = np.math.factorial(length-1)
    for ith in range(errors):
        denominator *= length + ith
        _hs *= hs
        _errors[ith] = sum(coefficients*_hs) / denominator
    return coefficients.tolist(), _errors


if __name__ == '__main__':
    hs = [0, -1, -2]
    order = 1
    c, e = deriving_finite_difference_approximations(hs, order, 3)
    print(c, e)

    hs = [-1, 0, 1]
    order = 2
    c, e = deriving_finite_difference_approximations(hs, order, 3)
    print(c, e)
```

1_2_sympy.py

```
#!/usr/bin/python3
import sympy


def deriving_finite_difference_approximations(u, x, h, hs, order=1, errors=1):
    '''See also `1_2.py`.
    '''
    # the method of undetermined coefficients
    length = len(hs)
    hs = sympy.Matrix([hs])
    _hs = sympy.ones(1, length)
    A = sympy.ones(length)
    for ith in range(1, length):
        _hs = _hs.multiply_elementwise(hs)
```

```python
        A[ith, :] = _hs
    b = sympy.zeros(length, 1)
    b[order] = sympy.factorial(order)
    coefficients = A.solve(b)
    # errors
    _errors = [None] * errors
    Du = u(x).diff(x, length-1)
    denominator = sympy.factorial(length-1)
    for ith in range(errors):
        denominator *= length + ith
        _hs = _hs.multiply_elementwise(hs)
        coefficient = coefficients.dot(_hs) / denominator
        Du = Du.diff()
        _errors[ith] = coefficient * h**(length+ith-order) * Du
    # finite difference approximations
    Du = sum(c*u(x+a*h) for c, a in zip(coefficients, hs))
    return Du/h**order, sum(_errors)


if __name__ == '__main__':
    u = sympy.Function('u')
    x, h = sympy.symbols('x, h')

    hs = [0, -1, -2]
    order = 1
    D1u, errors = deriving_finite_difference_approximations(u, x, h, hs, order, 1)
    sympy.pretty_print(D1u + errors)

    hs = [-1, 0, 1]
    order = 2
    D2u, errors = deriving_finite_difference_approximations(u, x, h, hs, order, 2)
    sympy.pretty_print(D2u + errors)

    hs = [-1, 0, 1, 2]
    order = 3
    D3u, errors = deriving_finite_difference_approximations(u, x, h, hs, order, 2)
    sympy.pretty_print(D3u + errors)

    hs = [-2, -1, 1, 2]
    order = 3
    D3u, errors = deriving_finite_difference_approximations(u, x, h, hs, order, 2)
    sympy.pretty_print(D3u + errors)
```