Schnittstellendokumentation

Habib Eser und Lukas Pauli

13. Mai 2022

Contents

1 Bedienungsanleitung

Diese Module bieten eine interaktive Möglichkeit, das 2D-Poisson-Problem mithilfe der Finite-Differenzen-Methode zu untersuchen. Der Fokus liegt auf dem Experimentierskript, welches die zentrale Komponente des Programms darstellt. Mit diesem Skript können Sie numerische Experimente durchführen, Parameter anpassen und Ergebnisse visualisieren.

Die verschiedenen Module des Programms arbeiten zusammen:

- `experiment_script.py`: Das Hauptskript, das alle anderen Module aufruft und die Benutzerinteraktion ermöglicht. Es steuert die Eingabe von Parametern, die Lösung des Poisson-Problems und die Visualisierung.
- `linear_solver.py`: Enthält die Logik zur numerischen Lösung eines LGS
- `poisson_problem2d.py`: Beinhaltet Methoden zur Analyse der numerischen und exakten Lösungen des Poisson Problems im Vergleich
- `block_matrix_2d.py`: Zentriert über die BlockMatrix A die man aus dem diskretisiertem Poisson-Problem herleiten kann.

Das Experimentierskript vereint diese Module, um Ihnen eine intuitive Plattform zur Durchführung numerischer Experimente zu bieten.

1.1 Anforderungen

Um das Modul zu verwenden, benötigen Sie:

- **Python-Installation** (Version 3.8 oder höher)
- **Erforderliche Python-Bibliotheken**: `numpy`, `scipy`, `matplotlib`. Installieren Sie diese mithilfe von:

```
pip install numpy scipy matplotlib
```

1.2 Start des Programms

1. **Herunterladen des Programms**: Laden Sie alle notwendigen Dateien des Moduls in ein lokales Verzeichnis.
2. **Öffnen der Hauptdatei**: Die Hauptdatei ist `experiment_script.py`. Öffnen Sie diese Datei in Ihrer bevorzugten Python-IDE oder im Terminal.
3. **Ausführen des Programms**: Starten Sie das Programm mit dem Befehl:

```
python experiments_lu.py
```

1.3 Benutzeroberfläche und Eingaben

Das Experimentierskript leitet Sie durch die Konfiguration und Durchführung eines numerischen Experiments. Folgen Sie diesen Schritten:

1. **Gittergröße definieren**: Geben Sie die gewünschte Gittergröße ein, z. B. `50` für ein 50x50-Gitter.
2. u **und** f **angeben**: Übergeben Sie die Funktionen n Python-syntaktischer Form, z. B.:

```
np.sin(np.pi * x[0]) * np.sin(np.pi * x[1])
```

3. **Optionale Parameter einstellen**:
   - $\alpha > 1$
   - κ
   - Maximale Anzahl an Intervallen max_n
   - Beispiel-Anzahl von Intervallen n
4. **Experiment starten**: Nachdem alle Eingaben gemacht wurden, startet das Skript den Berechnungsprozess und zeigt die Ergebnisse an.

Hinweise

Das Modul `experiment_lu.py` ist das Herzstück des Moduls. Es vereint die Funktionalitäten aller anderen Module und ermöglicht es, das Poisson-Problem flexibel zu untersuchen. Für komplexere Anpassungen können Sie die zugrundeliegenden Module (`linear_solver.py`, `poisson_problem2d.py`, `block_matrix_2d.py`) direkt bearbeiten.

2 linear_solver.py

Dieses Modul zielt darauf ab, lineare Gleichungssysteme mithilfe der LU-Zerlegung zu lösen. Es bietet eine Funktion welche mit Vorwärts- und Rückwärtssubstitution, das Gleichungssystem $A\mathbf{x} = \mathbf{b}$ effizient löst.

2.1 Methods

2.1.1 solve_lu

Diese Methode löst das lineare Gleichungssystem $A\mathbf{x} = \mathbf{b}$ mittels Vorwärts- und Rückwärtssubstitution, wobei die LU-Zerlegung $A = P \cdot L \cdot U$ verwendet wird. Man berechnet erst $P^T b = b'$. Als zweites löst man mit Vorwärtssubstitution $Ly = b'$ und als drittes löst man mit Rückwärtssubstitution $Ux = y$

Parameter:

- `p` (numpy.ndarray) : Permutationsmatrix der LU-Zerlegung.
- `l` (numpy.ndarray) : Untere Dreiecksmatrix mit Einheitsdiagonale der LU-Zerlegung.
- `u` (numpy.ndarray) : Obere Dreiecksmatrix der LU-Zerlegung.
- `b` (numpy.ndarray) : Vektor der rechten Seite des linearen Systems.

Returns:

- `x` (numpy.ndarray) : Lösung des linearen Gleichungs-Systems.

3 block_matrix_2d.py

Dieses Modul dient zur Konstruktion und Analyse von Blockmatrizen, die aus Finite-Differenzen-Approximationen des Laplace-Operators stammen. Es umfasst Funktionen wie die Darstellung spärlicher Matrizen, LU-Zerlegung, Bewertung der Sparsität und Visualisierung von Nicht-Null-Einträgen und Sparsität. Das Modul enthält folgende **Klasse**

- `BlockMatrix` : Repräsentiert Blockmatrizen mit Methoden zur spärlichen Speicherung, LU-Zerlegung, Bewertung der Sparsität und Berechnung der Konditionszahl.

3.1 Konstruktor

Der Konstruktor initialisiert die BlockMatrix mit der Anzahl der Intervalle $n \geq 2$. Der Wert von `n` wird in der Instanz gespeichert, und die Gesamtzahl der inneren Gitterpunkte wird berechnet. Zu Beginn ist die Matrix `None`.

```
def __init__(self, n):
    if n < 2:
        raise ValueError("n must be at least 2.")
    self.n = n
    # Total number of interior grid points
    self.gridsize = (n - 1) ** 2
    self.matrix = None
```

Input:

- `n` (`int`) : Die Anzahl der Intervalle in jeder Dimension. Muss mindestens 2 sein.

Raises:

- `ValueError` : Wird ausgelöst, wenn `n` kleiner als 2 ist.

3.2 Attribute

3.2.1 Nicht-statische Attribute

- `n` (`int`) : Die Anzahl der Intervalle in jeder Dimension.
- `gridsize` (`int`) : Die Gesamtzahl der inneren Gitterpunkte
- `matrix` (`numpy.ndarray` oder `None`) - Die Blockmatrix A induziert vom Poisson-Problem, die zu einem späteren Zeitpunkt erstellt wird. Zunächst ist sie `None`.

3.3 Methoden

3.3.1 get_sparse

Die Methode erstellt eine sparse Matrix für das Poisson-Problem, indem sie die Hauptdiagonale sowie die Off- und Side-Diagonalen der Blockmatrix A induziert von der diskretisierten Poisson-Gleichung, in einem Vektor speichert. Der Fall $n = 2$ wird besonders behandelt. Außerdem aktualisiert man `self.matrix` mit der Blockmatrix A.

Returns:

- `scipy.sparse.csr_matrix`: Die Blockmatrix A welche aus dem Poissonproblem ensteht im Sparse-Format.

3.3.2 get_lu

Diese Methode führt die LU-Zerlegung der dargestellten Matrix A durch. Anschließend wird die LU-Zerlegung der Matrix mit der Funktion `lu` von `scipy.linalg`
durchgeführt. Die Methode gibt die Permutationsmatrix, die untere Dreiecksmatrix und die obere Dreiecksmatrix der LU-Zerlegung zurück.

Returns:

- `p_matrix` (numpy.ndarray) : Die Permutationsmatrix der LU-Zerlegung.
- `l_matrix` (numpy.ndarray) : Die untere Dreiecksmatrix mit Einheitsdiagonalen der LU-Zerlegung.
- `u_matrix` (numpy.ndarray) : Die obere Dreiecksmatrix der LU-Zerlegung.

3.3.3 eval_sparsity_lu

Diese Methode berechnet die absolute und relative Anzahl der Nicht-Null-Elemente in der LU-Zerlegung der Matrix. Diese Gesamtzahl der Nicht-Null-Elemente in beiden Matrizen L und U (Hauptdiagonale von L ausgeschlossen) werden dabei zusammengerechnet. Die relative Zahl der Nicht-Null-Elemente wird als Verhältnis der Nicht-Null-Elemente zur Gesamtzahl der Elemente in der Matrix berechnet.

Returns:

- `int`: Die Gesamtzahl der Nicht-Null-Elemente in der LU-Zerlegung.
- `float` : Die relative Anzahl der Nicht-Null-Elemente in der LU-Zerlegung im Verhältnis zur Gesamtzahl der Elemente der Matrix.

3.3.4 get_cond

Diese Methode berechnet die Konditionszahl der dargestellten Matrix im Hinblick auf die Unendlich-Norm. Die Methode verwendet die Funktion `np.linalg.cond` aus der NumPy-Bibliothek, um die Konditionszahl der Matrix mit der Unendlich-Norm zu berechnen.

Returns:

- `float`: Die Konditionszahl der Matrix, berechnet mit der Unendlich-Norm.

3.3.5 plot_nonzeros_sparse_dense_vs_gridsize

Diese Methode erstellt ein Diagramm, das die Anzahl der Einträge der Matrix A als Funktion der Gittergröße N im Sparse-Format und im Dense-Format darstellt. Die Anzahl der Einträge wird sowohl für die spärliche als auch für die dichte Matrix auf einer logarithmischen Skala aufgetragen.

Parameters:

- `max_n` (`int`): Der maximale Wert von n, der für das Diagramm berücksichtigt wird.

3.3.6 plot_nonzeros_matrix_lu_vs_gridsize

Diese Methode erstellt ein Diagramm, das die Anzahl der nicht-null Einträge der Matrix A und ihrer LU-Zerlegung als Funktion der Gittergröße N darstellt. Für diese Zwecke wird die Methode `eval_sparsity_lu` verwendet.

Parameters:

- `max_n` (`int`): Der maximale Wert von n, der berücksichtigt wird.

3.3.7 plot_condition_number_vs_gridsize

Diese Methode erstellt einen Plot, das die Konditionszahlen der Matrix A und ihrer LU-Zerlegung (Matrixen L und U) als Funktion der Gittergröße N darstellt.

Parameters:

- `max_n` (`int`): Der maximale Wert von n, der berücksichtigt wird.

3.4 Nutzungshinweise und Hauptprogramm

Diese Methode demonstriert einige Methoden der `BlockMatrix`-Klasse. Es wird eine eine `BlockMatrix` mit $n = 4$ initialisiert. Dann wird die Matrix A ,die absolute und relative Anzahl der Nicht-Null-Einträge in der Matrix A Anzahl der Nicht-Null-Einträge in der LU-Zerlegung sowie deren relative Anzahl ausgegeben. Ebenso werden erzeugt Plots zu der Anzahl der Einträge, der Anzahl der Nicht-Null-Einträge und der Konditionszahlen in Bezug auf die Gittergröße N. Bei Interesse kann man n ändern.

```
def main():
    """This method demonstrates some methods of the blockmatrix class.
    There are plots to number of entries,
    nonzero entries and condition numbers vs N
```

```
"""
n = 4
block_matrix = BlockMatrix(n)
block_matrix.get_sparse()

# print A
print("Sparse Matrix:")
print(block_matrix.matrix)

# print absoulute and relative number of nonzero entries in A
num_nonzeros, relative_sparsity = block_matrix.eval_sparsity()
print(f"Number of non-zeros: {num_nonzeros}")
print(f"Relative number of non-zeros: {relative_sparsity}")

# Calculate and print LU sparsity of A
total_non_zeros_lu, relative_non_zeros_lu =
block_matrix.eval_sparsity_lu()
print(f"Number of non-zeros in LU decomposition:
{total_non_zeros_lu}")
print(f"Relative number of non-zeros in LU decomposition:
{relative_non_zeros_lu}")

# Plot the number of entries of A as a sparse and dense matrix
block_matrix.plot_nonzeros_sparse_dense_vs_gridsize(max_n=50)

# Plot the (number of non-zero entries) sparsity of A and LU
for comparison
block_matrix.plot_nonzeros_matrix_lu_vs_gridsize(max_n=30)

# Plot the condition of A
block_matrix.plot_condition_number_vs_gridsize(max_n=40)
```

4 poisson_problem2d.py

Dieses Modul zielt darauf ab, das 2D-Poisson-Problem mithilfe von Finite-Differenzen und LU-Zerlegung zu lösen. Es bietet Funktionen zum Einrichten, Lösen und Visualisieren des Poisson-Problems. Das Modul umfasst die Berechnung des Fehlers zwischen der numerischen Lösung und der exakten Lösung, die Visualisierung der exakten und numerischen Lösungen sowie deren Fehler und die Berechnung des vektors b aus der diskretisierten

Poisson-gleichung $Ax = b$.

4.1 Methoden

4.1.1 idx

Diese Methode berechnet die Gleichungsnummer im Poisson-Problem für einen gegebenen Diskretisierungspunkt.

Input:

- `nx` (list of int) : Koordinaten eines Diskretisierungspunkts, multipliziert mit n.
- `n` (int): Anzahl der Intervalle in jeder Dimension.

Returns:

- (int) Die Nummer der entsprechenden Gleichung im Poisson-Problem.

4.1.2 inv_idx

Diese Methode berechnet die Koordinaten eines Diskretisierungspunkts für eine gegebene Gleichungsnummer des Poisson-Problems.

Input:

- `m` (int): Nummer einer Gleichung im Poisson-Problem.
- `n` (int): Anzahl der Intervalle in jeder Dimension.

Returns:

- (list of int) Die zwei Koordinaten des entsprechenden Diskretisierungspunkts, multipliziert mit n.

4.1.3 rhs

Diese Methode berechnet den rechten Seitenvektor b für eine gegebene Funktion f im Poisson-Problem.

Input:

- `n` (int): Anzahl der Intervalle in jeder Dimension.
- `f` (callable): Repräsentiert die rechte Seite der Poisson-gleichung. Die Aufrufsignatur ist `f(x)`, wobei `x` ein `array_like` von `numpy` ist und der Rückgabewert ein Skalar ist.

Raises:

- `ValueError` : Wenn `n` kleiner als 2 ist.

Returns:

- `numpy.ndarray` : Der Vektor auf der rechten Seite des LGS, welches induziert wird von der diskretisierten Poisson-gleichung.

4.1.4 compute_error

Diese Methode berechnet den Fehler der numerischen Lösung des Poisson-Problems bezüglich der exakten Lösung, an den Gitterpunkten, in der Unendlich-Norm.

Input:

- `n` (int) : Anzahl der Intervalle in jeder Dimension.
- `hat_u` (numpy.ndarray) : Endliche Differenzen-Approximation der Lösung des Poisson-Problems, geordnet nach `idx`.
- `u` (callable) : Exakte Lösung des Poisson-Problems. Die Aufrufsignatur ist `u(x)`, wobei `x` ein `array_like` von `numpy` ist, das einen Punkt `(x[0], x[1])` darstellt.

Returns:

- `float` : Der maximale absolute Fehler an den Diskretisierungspunkten.

4.1.5 solve_poisson_problem

Diese Methode löst das das diskretisierte Poisson-Problem für ein gegebenes`n` und `f`, durch lösen der Gleichung $Ax = b$.

Input:

- `n` (int) : Anzahl der Intervalle in jeder Dimension.
- `f` (callable) : Rechte Seite Funktion in der Poisson-Gleichung

Returns:

- `numpy.ndarray` : Finite-Differenzen-Approximation der Lösung des Poisson-Problems, geordnet nach `idx`.

4.1.6 plot_poisson_problem

Diese Methode stellt die Lösung des Poisson-Problems für ein gegebenes `n`, exakte Lösung `u` und rechte Seite Funktion `f` grafisch dar. Im 3D-Plot werden die Graphen von der exakten Lösung und der numerischen Lösung gegenübergestellt.

Input:

- `n` (int) : Anzahl der Intervalle in jeder Dimension.
- `u` (callable) : Exakte Lösung des Poisson-Problems.
- `f` (callable) : Rechte Seite Funktion im Poisson-Problem.

4.1.7 plot_error_vs_N

Diese Methode stellt den Fehler der numerischen Lösung des Poisson-Problems als Funktion der Gittergröße N dar.

Input:

- `u` (callable) : Exakte Lösung des Poisson-Problems.
- `f` (callable) : Rechte Seite Funktion im Poisson-Problem.
- `max_n` (int) : Maximale Gittergröße (ausschließlich).

4.2 Nutzungshinweise und Hauptprogramm

In `poisson_problem2d.py` ist eine `main()`-Funktion implementiert, die zur Demonstration der wichtigsten beiden Methoden in diesem Modul dient. Sie wird nur ausgeführt, wenn `poisson_problem2d.py` direkt mittels `python3 poisson_problem2d.py` gestartet wird. Die Funktion u die verwendet wird, ist eine glatte Funktion, die für die festgelegten Werte gut das Approximationsverhalten der numerischen Lösung demonstriert. Man ersetze u und f mit anderen funktionen die das Poisson-Problem lösen, und wähle andere Werte für max_n und n, um mit den Methoden zu experimentieren.

```
def main():
    """Demonstration of the methods
    in this module on specific functions u, f.
    """
    kappa = 1
    def u(x):
        return x[0] * x[1] * np.sin(kappa * np.pi*x[0])
```

```
    * np.sin(kappa * np.pi*x[1])

def f(x):
    pi = np.pi
    x,y = x[0],x[1]
    laplacian =
    2 * kappa * pi * x * np.cos(kappa * pi * y)
    * np.sin(kappa * pi * x) + 2 * kappa * pi * y
    * np.cos(kappa * pi * x) * np.sin(kappa * pi * y)
    - 2 * kappa**2 * pi**2 * x * y * np.sin(kappa * pi * x)
    * np.sin(kappa * pi * y)
    return -laplacian

plot_error_vs_N(u,f,max_n = 30)
plot_poisson_problem(10, u, f)
```

5 experiments_lu.py

Dieses Modul bietet eine interaktive Umgebung zur Lösung des 2D-Poisson-Problems mithilfe der Finite-Differenzen-Methode. Es ermöglicht den Benutzern, eigene exakte Lösungen $u(x, y)$ und Quellterme $f(x, y)$ zu definieren. Neben der Lösung des Poisson-Problems werden Visualisierungen der Lösungen sowie Fehlerkonvergenz-Analysen bereitgestellt.

Hauptfunktionen

- **Benutzereingaben:** Benutzer können Parameter wie die Gitterauflösung n, die maximale Auflösung `max_n`, sowie eigene Funktionen für $u(x, y)$ und $f(x, y)$ angeben.
- **Interaktive Funktionserstellung:** Funktionen $u(x, y)$ und $f(x, y)$ werden aus benutzerdefinierten Python-Ausdrücken generiert.
- **Visualisierung:** Das Modul bietet:
  - Fehleranalyse durch Plots des Fehlers in Abhängigkeit von der Gittergröße.
  - Darstellung der berechneten Lösungen auf einem 2D-Gitter.
- **Beispiele:** Es stehen mehrere vorgefertigte Beispiele zur Verfügung, um typische Szenarien zu demonstrieren, wie z. B. Polynome oder exponentielle Funktionen.

Hauptkomponenten

`get_interactive_input()` Ermöglicht dem Benutzer, Eingaben wie Gittergröße, Parameter und Funktionen für $u(x, y)$ und $f(x, y)$ bereitzustellen.

`create_function_from_string()` Wandelt Python-Ausdrücke in ausführbare Funktionen um.

`plot_poisson_problem()` Visualisiert die Lösung des Poisson-Problems auf einem Gitter.

`plot_error_vs_N()` Zeigt den Fehlerverlauf bei variierender Gitterauflösung N.

`user_examples()` Enthält eine Sammlung von interaktiven Beispielproblemen, die unterschiedliche Szenarien und Funktionstypen abdecken.

Das Modul kann leicht erweitert werden, um weitere numerische Methoden oder Visualisierungsoptionen zu integrieren. Durch die offene Struktur können Benutzer neue Anwendungsfälle und Algorithmen einbinden.

5.1 methods

5.1.1 get_interactive_input

Diese Methode führt den Benutzer interaktiv durch die Eingabe von Parametern für die Lösung des Poisson-Problems.Genauer gesagt fordert sie den Benutzer auf, Eingaben für verschiedene Parameter des Poisson-Problems zu machen, einschließlich der Anzahl der Intervalle, der Obergrenze für die Fehleranalyse, der Werte von α und κ sowie der Funktionen $u(x_0, x_1)$ und $f(x_0, x_1)$, die die exakte Lösung des Poisson-Problems und die rechte Seite der Poisson-Gleichung definieren. Zu allen Eingaben wird auch eine Beispiel-Eingabe vorgeschlagen, die als Orientierung für den User dient.

Returns:

- `n` (int) : Anzahl der Intervalle für die diskrete Gitterdarstellung des Poisson-Problems.
- `max_n` (int) : Obergrenze für die Fehleranalyse im Zusammenhang mit der Gittergröße N.
- `alpha` (float) : Wert von α, der in den Funktionen verwendet wird.
- `kappa` (float) : Wert von κ, der in den Funktionen verwendet wird.

- `u_str` (string) : Der Python-Ausdruck für die exakte Lösung $u(x_0, x_1)$.
- `f_str` (string) : Der Python-Ausdruck für die rechte Seite $f(x_0, x_1)$.

5.1.2 create_function_from_string

Diese Methode konvertiert einen String in einen Python-Ausdruck, der als Funktion verwendet werden kann. Diese Funktion dient dem Zweck die Eingabe Funktionen des Users zurückgegeben von `get_interactive_input`, verwenden zu können. Genauer gesagt nimmt die Methode einen String, der eine mathematische Funktion beschreibt, und verwandelt ihn in eine Python-Funktion. Dabei werden zusätzliche Variablen, die in der Funktion verwendet werden, in das Evaluierungsumfeld aufgenommen. Dies ermöglicht es, den String-Ausdruck unter Verwendung von Variablen wie x und beliebigen zusätzlichen Variablen auszuführen.

Parameter:

- `func_str`: Ein String, der die mathematische Funktion beschreibt. Der Ausdruck wird später mit `eval` in eine Python-Funktion umgewandelt.
- `additional_vars`: Ein Dictionary, das zusätzliche Variablen enthält, die in der Funktion verwendet werden können. Die Variablen werden dem Evaluierungsumfeld hinzugefügt.

Returns:

- `callable` : Gibt eine Funktion zurück, die den Python-Ausdruck repräsentiert. Diese Funktion kann mit einem Wert für x aufgerufen werden, und der Ausdruck wird unter Verwendung des gegebenen Wertes und der zusätzlichen Variablen berechnet.

5.1.3 user_examples

Diese Methode bietet dem Benutzer standardisierte Beispiele, die er explorativ verwenden kann, um das Poisson-Problem zu untersuchen. Der Benutzer gibt die Parameter ein, und es werden verschiedene Beispiele für $u(x, y)$ und $f(x, y)$ bereitgestellt, um das Verhalten der Poisson-Gleichung zu visualisieren. Zudem wird im Terminal ausgegeben was die expliziten Formeln sind für u und f. Es werden Plots erstellt, um das Verhalten der Lösung und des Fehlers in Abhängigkeit von der Auflösung zu untersuchen.

Parameter:

- `max_n` (int) : Die maximale Zahl für n

- `n_` (int) : Die Anzahl der Intervalle für das Gitter
- `kappa_` (float) : Ein Parameter, der in den Funktionen $u(x, y)$ und $f(x, y)$ verwendet wird.
- `alpha_` (float) : Ein Parameter, der in einem der Beispiel-Ausdrücke verwendet wird.