Alexander Dominicus

Automatisiertes Assessment iterativer Verfahren im Mathematikbereich in Moodle

Zusammenfassung. Dieser Artikel beschreibt den Einsatz des Lern-Management-Systems Moodle in Kombination mit dem Plugin *Coderunner* beim Erlernen iterativer mathematischer Verfahren in der Hochschullehre. Es wird dargestellt, wie Coderunner Studierenden in Fächern wie Mathematik, Ingenieurwissenschaften und Informatik hilft durch direktes und präzises Feedback praktische Fähigkeiten im Umgang mit iterativen Algorithmen zu entwickeln. Insbesondere wird beschrieben wie Bewertungen individualisiert erfolgen können, um Lösungen iterativer Algorithmen qualitativ beurteilen zu können.

Einleitung

Im akademischen Kontext, insbesondere in Fächern wie Mathematik, Ingenieurswissenschaften und Informatik spielen iterative Verfahren eine zentrale Rolle. Diese Methoden, darunter das näherungsweise Lösen von Gleichungen, die Darstellung von Zahlen, die Lösung von Differentialgleichungen oder die Approximation von Funktionen durch Polynome, sind wesentliche Werkzeuge in der Hochschullehre. Der Erwerb von Fähigkeiten im Umgang mit diesen Verfahren ist essentiell, erfordert jedoch auch regelmäßige praktische Anwendungen, um ein tiefgreifendes Verständnis zu entwickeln. In den meisten Fällen verlangt dies eine intensive und zeitaufwändige Betreuung durch die Dozierenden: Es müssen Fragestellungen zur Bearbeitung durch die Studierenden entwickelt werden. Daraus resultiert ein studentischer Programmcode, der in der Regel manuell von den Dozierenden ausgeführt und bewertet wird. Motiviert durch [6] soll diese Form des Assessments vollständig automatisiert werden. Durch automatisiertes Assessment kann die Bewertung und Benotung erheblich vereinfacht und objektiver werden. Zudem wird entsprechendes Feedback sofort ausgegeben und motiviert die Studierenden gegebene Fragestellungen vollständig und korrekt zu bearbeiten ([8]). Eine Schlüsselrolle beim automatisierten Assessment spielt der Fragetyp Coderunner [7, 1] für das Moodle-System [2]. Coderunner ermöglicht es den Studierenden, einen Programmcode direkt im Moodle-System einzugeben und zu testen.

Coderunner

Coderunner ist ein innovativer Fragetyp innerhalb des Moodle-Systems, der speziell für die automatische Überprüfung von Programmcode konzipiert ist. Er wird in Moodle-Tests eingesetzt, um Studierenden eine praktische Programmiererfahrung zu ermöglichen. Eine Besonderheit von Coderunner ist die Unterstützung einer Vielzahl von Programmiersprachen, darunter Python3, C, C++, Java, PHP, JavaScript (NodeJS), SQL und Octave/Matlab. Der Einsatz des Fragetyps Coderunner läuft im einfachsten Fall folgendermaßen ab: Anhand der Fragestellung erstellen die Studierenden eine Funktion. Diese wird auf einem externen Server mit bestimmten, im Vorfeld definierten, Parametern (*Testfälle*) ausgeführt und die Ausgaben des Programmcodes an das Moodle-System zurückgegeben. Die erhaltenen Ausgaben werden mit den erwarteten Ausgaben der Testfälle verglichen und bewertet. Stimmen alle Ausgaben der studentischen Funktion mit den hinterlegten exakt überein, wird die Frage als "korrekt"gewertet. Andernfalls wird die Frage als "falsch"gewertet und die Studierenden müssen ihre Funktion entsprechend anpassen. Ein praktisches Beispiel wäre die folgende Python-Funktion, welche für eine Zahl n dessen Quadratzahl n^2 ausgibt:

```
def printsquare(n):
  print(n*n)
```

Diese Funktion lässt sich durch Testfälle überprüfen (`printsquare`(5), `printsquare`(-3), `printsquare`(0) usw.) und damit auch bewerten. Für die Bewertung iterativer mathematischer Näherungsverfahren, wie sie in diesem Artikel diskutiert werden, reicht der Vergleich auf exakte Übereinstimmung jedoch nicht aus. Dies lässt sich an folgender Musteraufgabe verdeutlichen:

Musteraufgabe. *Erstellen Sie eine Python-Funktion* appsqrt*(n,x), die auf Basis des Newton-Verfahrens eine Approximation an die Quadratwurzel der Zahl n berechnet. Als Startwert x_0 der Iteration wird die Zahl n gewählt. Die Funktion soll so erstellt werden, sodass für $\ell \geq 1$ und*

$x_\ell := \text{appsqrt}(n, \text{appsqrt}(n, x_{\ell-1}))$ *mit* $x_0 := \text{appsqrt}(n, n)$ *gilt:*

$$\begin{aligned} |\text{appsqrt}(n, x_0) - \sqrt{n}| &= |\text{appsqrt}(n, n) - \sqrt{n}| \\ &\geq |\text{appsqrt}(n, x_1) - \sqrt{n}| \\ &\geq |\text{appsqrt}(n, x_m) - \sqrt{n}| \\ &\geq |\text{appsqrt}(n, x_{m+1}) - \sqrt{n}| \end{aligned} \tag{1}$$

für alle $m \geq 1$*. D.h. durch iterierte Funktionsanwendung wird die Genauigkeit der Berechnung verbessert.*

Die folgende Funktion beschreibt eine mögliche Lösung der Musteraufgabe:

```
def appsqrt(n,x):
  new_approx=0.5*(x+n/x)
  return new_approx
```

In diesem Fall besteht die Aufgabe nicht darin eine bestimmte Ausgabe zu erzeugen, die mit einer hinterlegten Lösung verglichen wird. Stattdessen soll der studentische Programmcode qualitativ im Hinblick auf (1) getestet werden. Um diese Aufgabe im Moodle-System darzustellen, muss eine eigene Bewertungsvorlage einer Coderunner-Frage erstellt werden.

Eigene Berwertungsvorlagen in Coderunner entwickeln

Grundlage der Bewertung einer Coderunner-Frage ist die hinterlegte Bewertungsvorlage. Diese ist im hohen Maße adaptierbar, wodurch eine präzise Beurteilung des studentischen Programmcodes sichergestellt werden kann. Dies wollen wird hier genutzt, um eine qualitative Bewertung der Musteraufgabe auf Basis von (1) sicherzustellen. Um diese Anpassung vorzunehmen, navigiert man im Moodle-System zum Bereich "Frage bearbeiten". Unter dem Reiter *Anpassung* findet man die Option *Anpassen*, welche den Zugang zu weiteren Einstellungen ermöglicht. Nach der Auswahl der Option *Anpassen* öffnet sich ein Editor, in dem ein selbst erstelltes Python-Skript hinterlegt werden kann, das für die Bewertung und Benotung der eingereichten Lösungen verwendet wird. Dies lässt eine flexible und präzise Beurteilung zu, da das Bewertungsskript speziell auf die Anforderungen der jeweiligen Aufgabe zugeschnitten werden kann. Auf diese

Weise lässt sich sicherstellen, dass die Bewertung nicht nur auf der Grundlage einer einfachen Übereinstimmung erfolgt, sondern auch komplexere Aspekte der Programmlösung berücksichtigt werden können. Im Fall der Musteraufgabe wird die folgende Bewertungsvorlage erstellt:

```
import math
import random
from random import uniform
NUM_ITER = 16
{{ STUDENT_ANSWER | e('py') }}
ok = True
TESTS=21
TOLERANCE=10e-6
for test in range(-1,TESTS):
    if test > TESTS:
        break
    n = random.randint(1, 10000)
    expected = math.sqrt(n)
    startvalue=n
    lasterror=abs(startvalue-expected)
    for i in range(NUM_ITER):
        stud_answer = appsqrt(n, startvalue)
        startvalue = stud_answer
        newerror=abs(stud_answer-expected)
        if lasterror<newerror:
            print("""Der Algorithmus nähert
                  sich nicht der exakten Lösung an!""")
            ok = False
            break
        lasterror = newerror
    if abs(expected - stud_answer) <= TOLERANCE:
        print("Gewünschte Toleranz erreicht!")
        print("n=",n)
        print("Anzahl Iterationen: {}, Fehler: {}".format(i, newerror))
    else:
        print("Gewünschte Toleranz nicht erreicht!")
        print("Fehler= ", newerror)
        ok = False
        break
if ok:
    print("Alle Tests bestanden!")
```

Um die Lesbarkeit zu erhöhen wurde das obige Template gekürzt. Das vollständige Python-Skript befindet sich Aufgabenpool des DigiTeach-Instituts [3]. Neben dieser Aufgabe sind dort auch weitere Beispiele zur individuellen Bewertung vorhanden.

Im obigen Bewertungsskript werden als erstes alle erforderlichen Pakete

importiert, die studentische Lösung mit Hilfe der Twig Template-Engine ausgelesen sowie alle benötigten Konstanten gesetzt. Daraufhin beginnt die erste Schleife, welche eine zuvor festgelegte Anzahl an Testfällen durchläuft (20 Testfälle): Es werden eine zufällige natürliche Zahl zwischen 1 und 10000 gewählt, die entsprechende Wurzel als Referenzlösung berechnet, der Startwert gesetzt und der Anfangsfehler berechnet. Anschließend wird in einer weiteren Schleife das Newton-Verfahren (die studentische Funktion) gemäß der vorgegebenen Anzahl an Iterationen durchgeführt. Dabei wird bei jedem Iterationsschritt geprüft, ob die neue Lösung näher (verglichen mit der Lösung aus dem vorangegangenen Iterationsschritt) an der Referenzlösung liegt. Falls der Fehler sich vergrößert hat, wird die Bewertung abgebrochen und die studentische Lösung nicht gewertet. Ist der Fehler im Iterationsschritt kleiner geworden, wird die nächste Iteration durchgeführt. Sind alle Iterationsschritte durchlaufen, wird geprüft, ob die Newton-Iteration unter einer vorgegebenen Schranke liegt. Wenn ja, wird den Studierenden ein Feedback zum Testfall ausgegeben: Die Testzahl n, die Zahl der Iterationen und der verbleibender Fehler. Am Ende wird den Studierenden eine finales Feedback zum erfolgreichen Abschluss der Testfälle ausgegeben.

Ausblick

Diese Arbeit beschreibt den Einsatz automatisierter Assessments im Mathematikbereich. Analog zu [6] sollte eine Untersuchung zur Verbesserung der Studienleistung und/oder Verbesserung der Lernmotivation durchgeführt werden. Eine weitere Einsatzmöglichkeit von Coderunner ist das Schaffen freier Lernumgebungen für Studierende innerhalb des Moodle-Systems. Möglich wird dies durch Nutzung des angepassten Fragetyps *Sandpit* ([4]), welcher nicht auf einer Bewertung des studentischen Programmcodes basiert, sondern lediglich zu dessen Ausführung dient. Somit können Dozierende Lernumgebungen vergleichbar mit einem Jupyter-Notebook innerhalb des Moodle-Systems für die Studierenden bereitstellen. Ein weiterer Schritt wäre dort die Bereitstellung von spezialisierten Software-Tools. Als Beispiel sind hier Finite-Elemente-Toolboxen (vgl. [5]) zu nennen, welche direkt im Moodle-System von den Studierenden verwendet werden könnten. Auf diese Weise können Studierende auf eine interaktive und nutzerfreundliche Art mit solchen Methoden vertraut gemacht werden. Die Installation spezieller Software durch die Studierenden entfällt

somit, da sich in diesem Fall die Testumgebungen vollständig im Moodle-System befinden. Daher trägt CodeRunner nicht nur zur Verbesserung der Programmierfähigkeiten bei, sondern erweitert auch das Verständnis für komplexe mathematische Methoden und deren Anwendung.

Literatur

[1] URL: `https://coderunner.org.nz/`.

[2] URL: `https://moodle.org/`.

[3] URL: `https://www.hochschule-bochum.de/digiteach/materialien-downloads/`.

[4] URL: `https://coderunner.org.nz/mod/page/view.php?id=558`.

[5] URL: `https://github.com/kinnala/scikit-fem`.

[6] Florian Horn, Daniel Schiffner und Detlef Krömker. »Akzeptanz der Nutzung von automatisiertem Assessment im Rahmen einer virtuellen Vorlesung«. In: *Proceedings of the Fifth Workshop 'Automatische Bewertung von Programmieraufgaben' (ABP 2021), virtual event, October 28-29, 2021*. 2021.

[7] Richard Lobb und Jenny Harlow. »Coderunner: A tool for assessing computer programming skills«. In: *ACM Inroads* 7.1 (2016), S. 47–51.

[8] Zahid Ullah u. a. »The effect of automatic assessment on novice programming: Strengths and limitations of existing systems«. In: *Computer Applications in Engineering Education* 26.6 (2018), S. 2328–2341.

Autor

Dr. Alexander Dominicus
DigiTeach-Institut
Hochschule Bochum
Am Hochschulcampus 1
D-44801 Bochum
E-Mail: alexander.dominicus@hs-bochum.de