FH AACHEN
UNIVERSITY OF APPLIED SCIENCES

Fachbereich Elektrotechnik
und Informationstechnik

Prof. Georg Hoever

SS 2025

1. Praktikum zur Höhere Mathematik 2 für (Wirtschafts-)Informatik

Ziel dieses Praktikums ist eine Implementierung des Gradientenverfahrens mit Schrittweitensteuerung.

Hinweis: Die Implementierung der Methoden zu Aufgabe 1 sowie die Implementierung der Funktion zu Aufgabe 2 und das Gradientenverfahren zu Aufgabe 3 sollen alle zusammen in **einer** cpp-Datei enthalten sein.

1. Aufgabe

Um bequem mit Vektoren $\vec{x} \in \mathbb{R}^n$ arbeiten zu können, soll eine Klasse `CMyVektor` implementiert werden:

- Überlegen Sie sich, durch welche(n) Datentyp(en) Sie die Informationen speichern, z.B. die Dimension als Integer und die Werte in einem `double`-Array; Sie können auch die Standardklasse `vector<double>` der C++-Standard-Template-Library nutzen.

  Implementieren Sie die Informationen als private Attribute.

- Implementieren Sie (public-)Methoden, um
  - einen Vektor einer bestimmten Dimension anzulegen,
  - die Dimension eines Vektors zurückzugeben,
  - eine bestimmte Komponente des Vektors zu setzen,
  - eine bestimme Komponente des Vektors zurückzugeben.

  Tipp: Sie können beispielsweise elegant den Indexoperator `[]` oder Klammeroperator `()` überladen.

- Implementieren Sie eine (public-)Methode, die die Länge des Vektors zurückgibt.

Implementieren Sie ferner zwei überladene Operator-Funktionen

```
CMyVektor operator+(CMyVektor a, CMyVektor b)
CMyVektor operator*(double lambda, CMyVektor a),
```

die eine Vektor-Addition und eine skalare Multiplikation realisieren.

2. Aufgabe

Zu einer Funktion $f : \mathbb{R}^n \to \mathbb{R}$ soll der Gradient an einer Stelle $\vec{x} \in \mathbb{R}^n$ berechnet werden:

- Implementieren Sie eine Funktion

  ```
  CMyVektor gradient(CMyVektor x, double (*funktion)(CMyVektor x)),
  ```

  der man im ersten Parameter die Stelle $\vec{x}$ und im zweiten Parameter die Funktion f als Funktionspointer übergibt, und die den Gradienten $\vec{g} = \operatorname{grad} f(\vec{x})$ numerisch durch

  $$g_i = \frac{f(x_1, \ldots, x_{i-1}, x_i + h, x_{i+1} \ldots, x_n) - f(x_1, \ldots, x_n)}{h}$$

  zu festem $h = 10^{-8}$ berechnet.

3. Aufgabe

Zu einer Funktion $f : \mathbb{R}^n \to \mathbb{R}$ soll ausgehend von einer Stelle $\vec{x} \in \mathbb{R}^n$ das Gradientenverfahren zur Maximierung von f mit der folgenden Schrittweitensteuerung durchgeführt werden:

1. Man geht testweise einen Schritt in Richtung des Gradienten mit der aktuellen Schrittweite.
2. Ist der Funktionswert nach dem Testschritt von 1. kleiner oder gleich dem aktuellen Funktionswert, so halbiert man die Schrittweite.

   Dies führt man solange durch, bis man eine Stelle mit größerem Funktionswert gefunden hat.
3. Ist der Funktionswert nach dem Testschritt von 1. größer als der aktuelle Funktionswert, so testet man – um gegebenenfalls schneller voran zu kommen – eine doppelt so große Schrittweite und arbeitet mit der Schrittweite weiter, die den größeren Funktionswert liefert.

Dieses Verfahren soll solange durchgführt werden, bis $\|\operatorname{grad} f(\vec{x})\| < 10^{-5}$ ist, oder bis 25 Schritte gemacht wurden.

- Implementieren Sie das entsprechende Verfahren als Funktion.

  Nutzen Sie wieder einen Funktions-Pointer zur Angabe der zu maximierenden Funktion. Neben der Startstelle $\vec{x}$ soll die Start-Schrittweite λ optionales Argument mit default-Wert `1.0` sein.
- Testen Sie das Verfahren an den folgenden Beispielen:
  - $f : \mathbb{R}^2 \to \mathbb{R}$, $f(x, y) = \sin(xy) + \sin x + \cos y$, Startstelle $\vec{x} = (0.2, -2.1)^T$, default-Start-Schrittweite,
  - $g : \mathbb{R}^3 \to \mathbb{R}$, $g(x_1, x_2, x_3) = -(2x_1^2 - 2x_1x_2 + x_2^2 + x_3^2 - 2x_1 - 4x_3)$, Startstelle $\vec{x} = (0, 0, 0)^T$, Start-Schrittweite $\lambda = 0.1$.

4. Aufgabe (optional, passend zu Bonus-e-Test 3b)

Beim Bonus-e-Test 3b, Aufgabe 3, werden Ihnen sieben Punkte

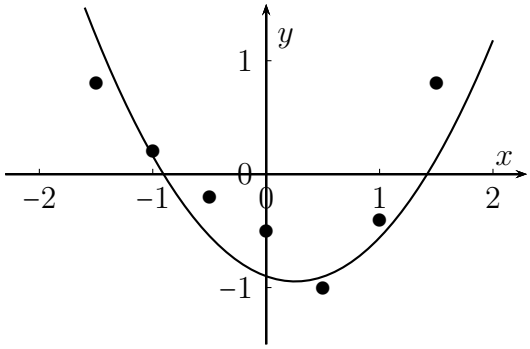
$$(-1.5, y_0), \quad (-1, y_1), \quad (-0.5, y_2), \quad (0, y_3), \quad (0.5, y_4), \quad (1, y_5), \quad (1.5, y_6)$$

vorgegeben.

Gesucht ist eine Ausgleichs-Parabel $p(x)$ zu diesen Punkten, also eine Parabel, so dass mit $x_0 = -1.5$, $x_1 = -1$, …, $x_6 = 1.5$ die Summe der quadratischen Abstände

$$s = \sum_{i=0}^{6} |p(x_i) - y_i|^2$$

minimal ist.



- Auf meiner Internetseite finden Sie eine Excel-Datei „Ausgleichs-Parabel.xls“.

  Laden Sie sich die Datei herunter und tragen Sie die y_k-Werte aus Ihrem Bonus-e-Test 3b in die gelb hinterlegten Zellen ein.

  In der Datei können Sie mit Schiebereglern die Parameter a, b und c einstellen.

  Angezeigt werden Ihre Punkte und die eingestellte Parabel zu $p(x) = ax^2 + bx + c$; berechnet wird die Summe der quadratischen Abweichungen s.

- Berechnen Sie mit dem Gradientenverfahren ausgehend vom Startpunkt $(0, 0, 0)$ optimale Parameterwerte für die Ausgleichsparabel zu Ihren Punkten aus dem Bonus-e-Test 3b.

  Sie können dazu eine eigene Funktion schreiben oder die c++-Funktion „abweichung.cpp“ auf meiner Internetseite nutzen, die die Summe der quadratischen Abstände zwischen vorgegebenen Punkten und einer Parabel berechnet.

  Hinweise:

  - Falls Sie die Funktion auf meiner Internetseite nutzen und in Ihr Projekt integrieren, so tragen Sie dort an der angegebenen Stelle die y_k-Werte aus Ihrem Bonus-e-Test 3b ein. Beachten Sie, dass die Funktion als Zugriff auf die Elemente von `CMyVektor` den Klammeroperator nutzt. Ggf. müssen Sie die Zugriffsfunktion entsprechend Ihrer `CMyVektor`-Klasse anpassen.
  - Die Summe der quadratischen Abstände soll *minimiert* werden. Dazu können Sie Ihr Gradientenverfahren so modifizieren, dass es automatisch minimiert. Alternativ können Sie $-s$ maximieren.

- Stellen Sie die Schieberegler der Excel-Datei auf die berechneten Werte ein, und überzeugen Sie sich, dass eine gute Ausgleichs-Parabel erzeugt wird.

Hinweis:

Falls Ihre Optimierung wegen der begrenzten Schrittanzahl endet, können Sie die Schrittanzahl erhöhen oder schauen, ob die berechneten Parameter schon eine gute Ausgleichs-Parabel erzeugen und in der Mumie akzeptiert werden.