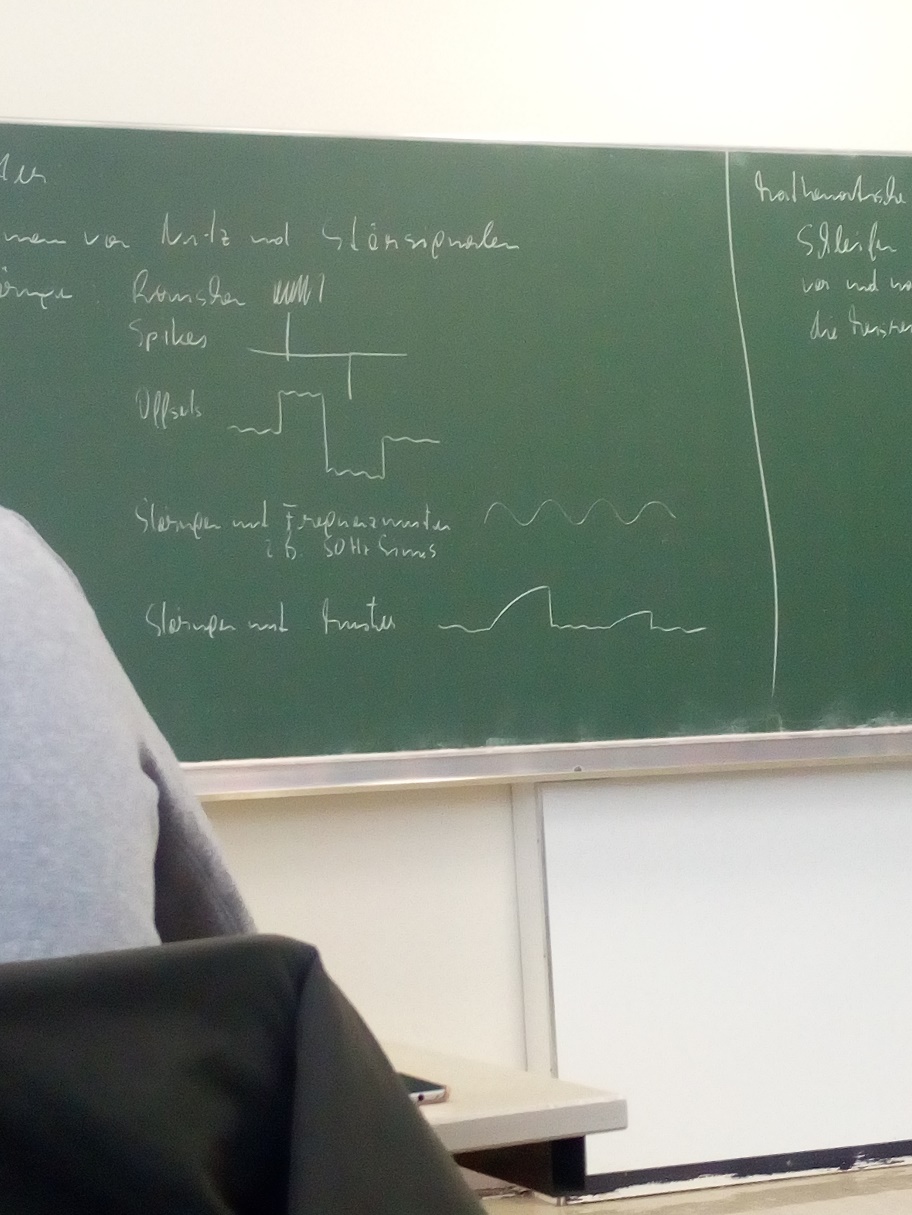
Filtern

Fensterbreite = Fenster: Anzahl der betrachteten Messwerte  
Vorgänger … Vergangenheit  
Nachfolger … Zukunft

m[i]…Messwerte  
anz…Anzahl der Messwerte  
gm[i]…geordnete Messwerte  
anzG…Anzahl der Gewichte  
g[i]…Gewichte

**Störungen:**

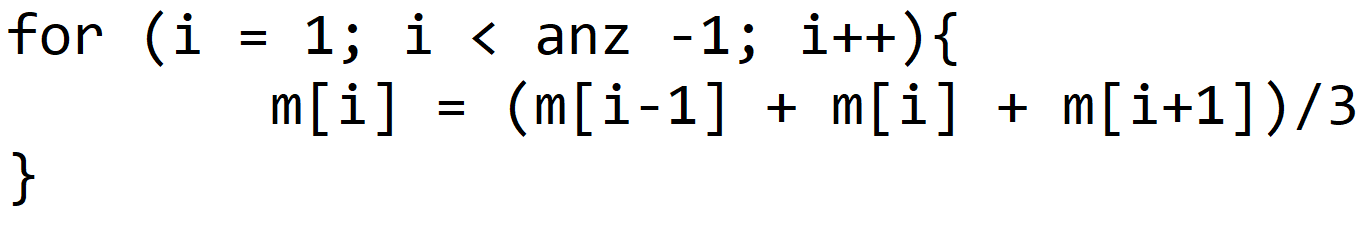


* Rauschen…keine fixe Frequenz und Amplitude
* Störungen mit bestimmter Frequenz (periodische Störsignale) (sinusschwingungn)
* Störungen mit bestimmten Muster  
  zb in der Elektronik 🡪 Kondensator Ladung und Entladung
* Ausreißer (Spikes)
* Versetzungen (Offsets)

1. Mittelwert (gegen Rauschen)

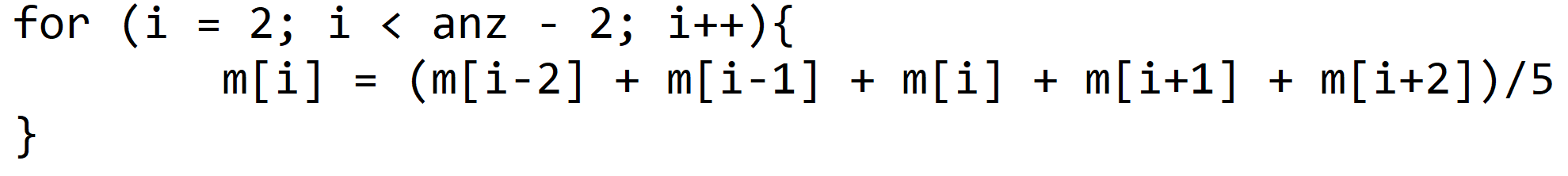
Echtzeitfähig:   
Phasentreu:

3. Punkt Mittelwert



5. Punkt Mittelwert

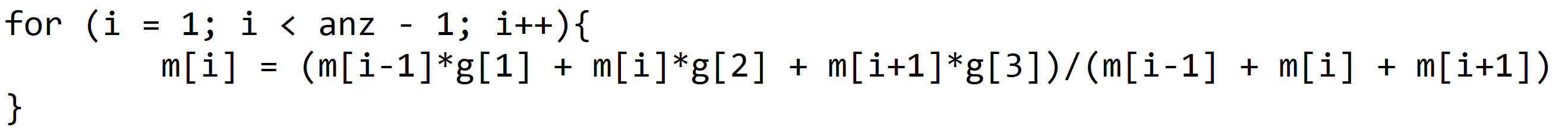
„verschmiert“ die Messwerte flacher aber offensichtlich richtige Messwerte werden verfälscht



1. Dreiecks

Echtzeitfähig: nein  
Phasentreu: ja

Erweiterung zu Mittelwertfilter mit Gewichten: Gewichte in Form eines Dreiecks 121 oder 12321 oder13531  
Nachbarn sind „weniger“ wichtig 🡪 glättet weniger als der Mittelwertfilter

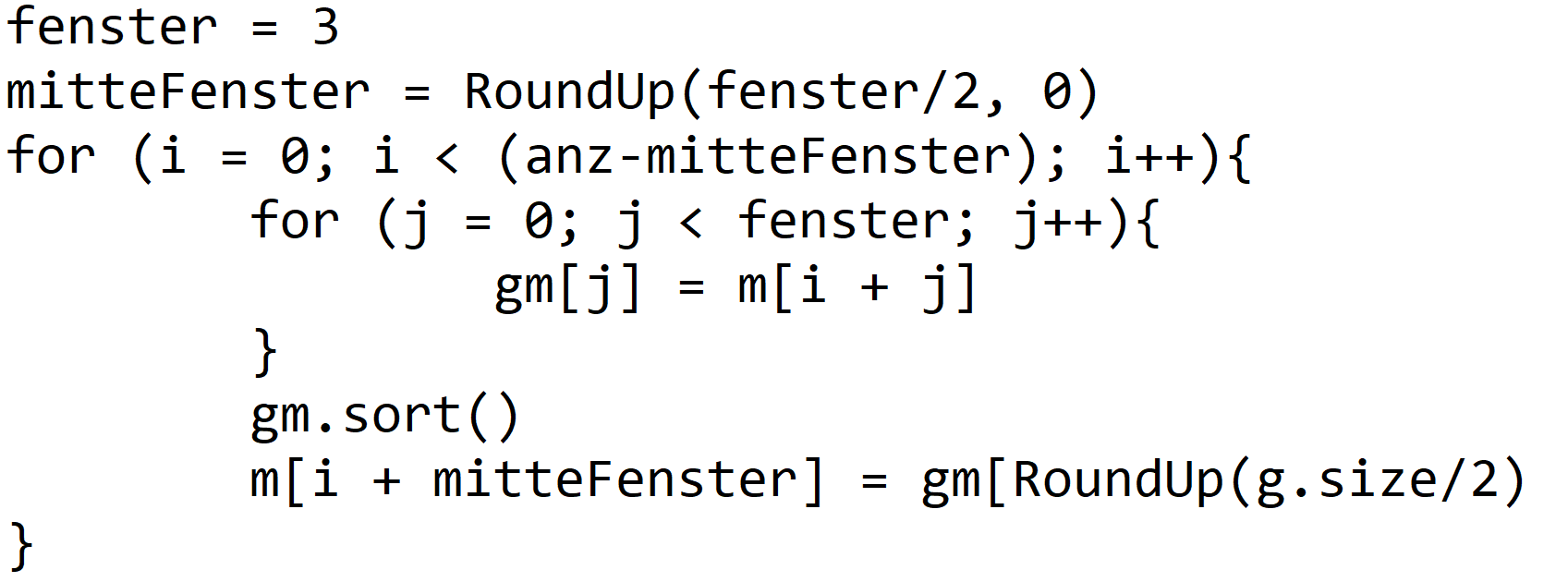


1. Median (gegen Spikes)

Echtzeitfähig: Nein  
Phasentreu: Nein

Fensterbreite >= 3 🡪 Fensterbreite = (Anzahl der Spikes im Fenster \* 2) + 1

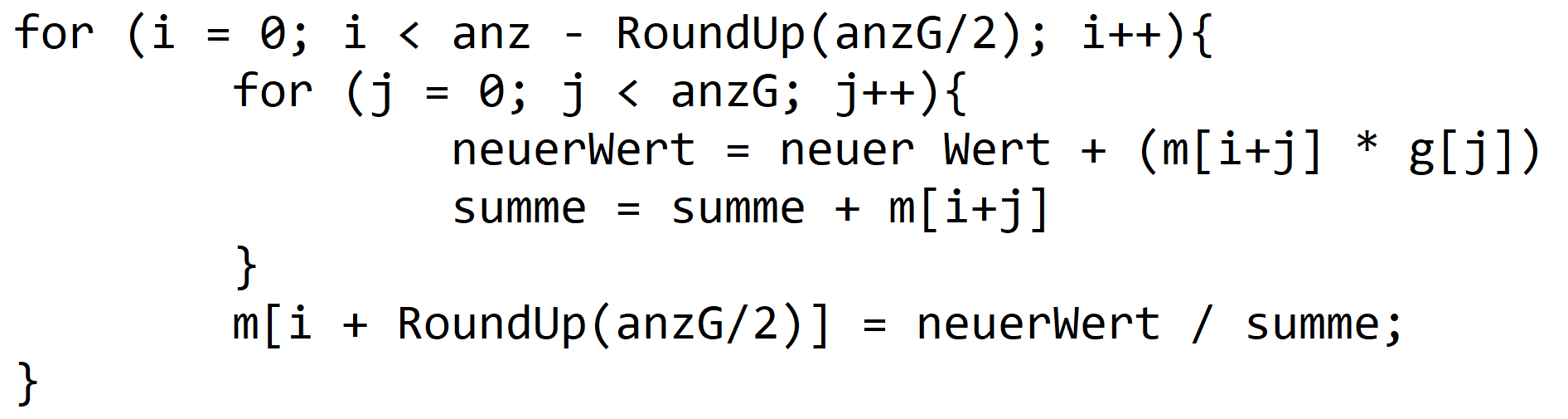
* +: kein Vorwissen über die Art der Spikes nötig
* -: langsam v^4
* Glättet stark, aber Werte sind Werte, die in der Messreihe vorkommen



1. allgemein gewichteter Filter

Echtzeitfähig:  
Phasentreu:

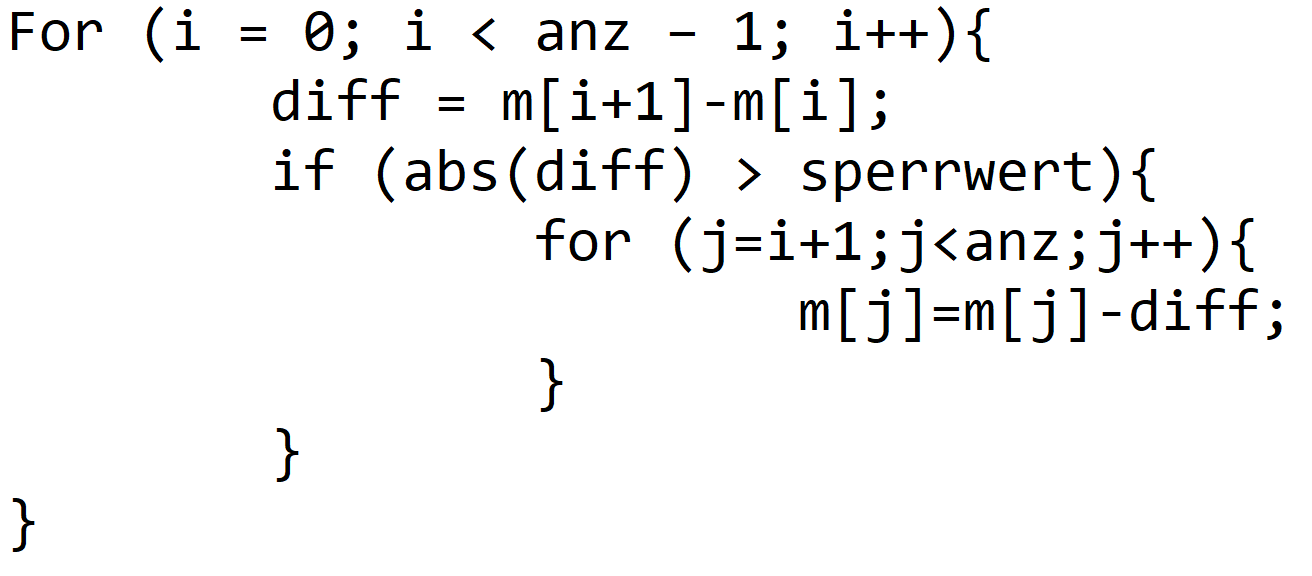
Gewichte Anzahl: min 1, max Anzahl der Messwerte – 2



1. Differenzen

Echtzeitfähig:  
Phasentreu:

Sperrwert: maximale Differenz, die zwei Werte auseinander sein dürfen



1. Algorithmus der Mustersuche
2. Begriffe

Echtzeitfähig

Wenn Wert gemessen wird, kann man ihn gleich filtern 🡪 keine Werte aus der Zukunft

Phasentreu

Maximum & Minimum an der gleichen Stelle

Datenübertragung

1. Aufbau eines einfachen json Strings

JavaScript Object Notation 🡪 key-value pairs

Keys: string  
Value:

* a string
* a number
* an object (JSON object)
* an array
* a Boolean
* null

{“key”:value}

1. Theorie sockets (tcp und datagramm)

Socket: Schnittstelle zwischen Anwendung & Netzwerk  
Datenaustausch zwischen zwei Netzwerkclients (= IP-Adressen) 🡪 zusätzlich Port festgelegt (! Well-Known Ports)

Zwei Arten (unterschiedlich nach verwendetem Protokoll)

* **Stream-Sockets (TCP): wie telefonieren**  
  Server 🡪 öffnet socket 🡪 Listener auf Port & eventuell IP Adresse vom Client  
  Client 🡪 öffnet socket 🡪 connect Anfrage zu Server IP & Port

Server 🡪 accept 🡪 Socketpaar erzeugen🡪 read & write über Socketpaar bis einer die Verbindung beendet

* **Datagramm-sockets(udp): wie SMS versenden**Client 🡪 erzeugt Socket mit Zieladresse & Port & schickt Daten ab. Fertig!

Alle Teilnehmer müssen Protokoll kennen & sich daran halten

1. Programmieren asynchronen TCP Server und TCP Client
2. Programmieren eines charts in C#
3. Programmieren Datenübertragung per USB von Arduino zu einer C# Anwendung am PC

Messtechnik:

1. Schaltungsaufbau eines Temperatursensors mittels Temperaturwiderstand und Spannungsteiler am Arduino

Begriffe

Well-Known Ports = niedrige Ports meist belegt

Protokoll: Art und Weise des Datenverkehrs 🡪 Verbindungsaufbau, Paketarten, …

Socketpaar: ServerIP, Port, ClientIP, Port