Computerphysik Programmiertutorial 4b

Prof. Dr. Matteo Rizzi und Dr. Markus Schmitt - Institut für Theoretische Physik, Universität zu Köln

ILIAS: https://www.ilias.uni-koeln.de/ilias/goto_uk_crs_3862489.html

Github: https://github.com/markusschmitt/compphys2021

Inhalt dieses Notebooks: Numerische Integration, Automatisches Differenzieren, Tabulare Datenstruktur

Numerische Integration: `Cubature`

Algorithmen zur numerischen Integration sind in verschiedenen Julia Paketen implementiert.

Ein Beispiel ist das Paket `Cubature`, das hier dokumentiert ist. `Cubature` bietet verschiedene Funktionen zur adaptiven numerischen Integration, z.B. `hquadrature` mit dem folgenden Interface:

```
(val,err) = hquadrature(f::Function, xmin::Real, xmax::Real; reltol=1e-8, abstol=0, maxevals=0)
```

In [1]:
```
using Cubature
using PyPlot
```

In [2]:
```
f(x)=3x+4x^2
F(x)=1.5x^2+(4.0/3.0)x^3
```

Out[2]:
```
F (generic function with 1 method)
```

Berechne das Integral $\int_{-1}^{3} f(x)dx$

In [9]:
```
int1 = hquadrature(f, -1, 3)
```

Out[9]:
```
(49.33333333333333, 5.541938576750177e-13)
```

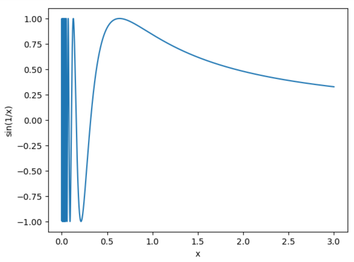
In [10]:
```
abs(F(3.0)-F(-1.0)-int1[1])
```

Out[10]:
```
7.105427357601002e-15
```

Adaptive Integrationsalgorithmen sind wichtig um z.B. Integrale von schnell oszillierenden Funktionen zu berechnen. Schauen wir uns als Beispiel $f(x) = \sin(1/x)$ an:

In [11]:
```
f(x)=sin(1.0/x)

interval=1e-10:1e-4:3
plot(interval,f.(interval))
xlabel("x")
ylabel("sin(1/x)")
```



Out[11]:
```
PyObject Text(24.0, 0.5, 'sin(1/x)')
```

In [12]:
```
hquadrature(f,1e-10,3,reltol=1e-8)
```

Out[12]:
```
(1.530630208145549, 1.5306295241461796e-8)
```

In diesem Fall spielen die weiteren Parameter eine Rolle, die als **keyword arguments** übergeben werden:

In [13]:
```
for lognsteps in [1,2,3,4,5,6,7,8]
    println(hquadrature(f,1e-10,3,reltol=1e-8,maxevals=10^lognsteps))
end
```

```
(1.5292609890154742, 0.7033433784755834)
(1.5873441061256683, 0.21321921239979105)
(1.5304894994405565, 0.003684948595490879)
(1.5306261409341617, 0.0002638564024441723)
(1.5306299105290975, 2.7436398775051424e-5)
(1.5306302066152184, 2.6941256664511534e-6)
(1.5306302070041407, 2.5145184161060763e-7)
(1.5306302081799639, 2.5140955411619077e-8)
```

Automatisches Differenzieren: `Zygote`

Durch **automatische Differenzierung** können wir die Ableitung beliebiger Funktionen berechnen, siehe Bonusaufgabe in dieser Woche. Ein Paket, das Funktionalität für automatisches Differenzieren bereitstellt ist `Zygote`, das hier dokumentiert ist.

In [14]:
```
using Zygote
```

```
┌ Warning: Error requiring `Colors` from `Zygote`
│   exception = (UndefVarError(:_parameter_upper_bound), Union{Ptr{Nothing}, Base.InterpreterIP}[Ptr{Nothing} @0x00000010a9f286f, Ptr{Nothing} @0x00000010aa8573f, Ptr{Nothing} @0x00000011351140, Ptr{Nothing} @0x00000010aa709ef, Ptr{Nothing} @0x00000010aa8773f, Ptr{Nothing} @0x00000010aa86094, Ptr{Nothing} @0x00000010aa8640c, Base.InterpreterIP in top-level CodeInfo for Zygote at statement 17, Ptr{Nothing} @0x00000010aaa0d44, Ptr{Nothing} @0x00000010aaa1634, Ptr{Nothing} @0x00000019e98d26d, Ptr{Nothing} @0x00000019e98d28c, Ptr{Nothing} @0x00000010aa709ef, Ptr{Nothing} @0x00000010aa8773f, Ptr{Nothing} @0x00000010aa86094, Ptr{Nothing} @0x00000010aa862be, Ptr{Nothing} @0x00000010aa85b2c, Base.InterpreterIP in MethodInstance for err(::Any, ::Module, ::String) at statement 2, Ptr{Nothing} @0x00000019e98d1c7, Ptr{Nothing} @0x00000019e98d1dc, Ptr{Nothing} @0x00000010aa709ef, Ptr{Nothing} @0x00000010aa8773f, Ptr{Nothing} @0x00000010aa85e13, Ptr{Nothing} @0x00000010aa862be, Ptr{Nothing} @0x00000010aa85b2c, Base.InterpreterIP in MethodInstance for withpath(::Any, ::String) at statement 10, Ptr{Nothing} @0x00000019e98d10c, Ptr{Nothing} @0x00000019e98d13c, Ptr{Nothing} @0x00000010aa709ef, Ptr{Nothing} @0x00000010aa8773f, Ptr{Nothing} @0x00000010aa86094, Ptr{Nothing} @0x00000010aa85b2c, Base.InterpreterIP in MethodInstance for listenpkg(::Any, ::Base.PkgId) at statement 3, Ptr{Nothing} @0x00000019e98cfe4, Ptr{Nothing} @0x00000019e98cfbc, Ptr{Nothing} @0x00000010aa709ef, Ptr{Nothing} @0x00000019e98e297, Ptr{Nothing} @0x00000019e98e2fe, Ptr{Nothing} @0x00000010aa709ef, Ptr{Nothing} @0x00000010aa9efe3, Ptr{Nothing} @0x00000010aa90b97, Ptr{Nothing} @0x00000001224d2424, Ptr{Nothing} @0x00000001224d37e3, Ptr{Nothing} @0x00000001224d5efa, Ptr{Nothing} @0x00000001224d8693, Ptr{Nothing} @0x00000001224d94bd, Ptr{Nothing} @0x00000010aa709ef, Ptr{Nothing} @0x00000010aa146c, Ptr{Nothing} @0x00000010aaa01d3, Ptr{Nothing} @0x00000010aaa0b04, Ptr{Nothing} @0x00000010aaa1634, Ptr{Nothing} @0x00000012252257c, Ptr{Nothing} @0x00000001225229e7, Ptr{Nothing} @0x00000010aa709ef, Ptr{Nothing} @0x00000012256849, Ptr{Nothing} @0x00000010aa709ef, Ptr{Nothing} @0x00000010aa7d269, Ptr{Nothing} @0x0000000122510e70, Ptr{Nothing} @0x00000001225111d2, Ptr{Nothing} @0x00000012251 1ec, Ptr{Nothing} @0x00000010aa709ef, Ptr{Nothing} @0x00000010aa8ba7d])
└ @ Requires /Users/markus/.julia/packages/Requires/7Ncym/src/require.jl:49
```

In [15]:
```
f(x)=3x+4x^2
f_prime(x)=3+8x
```

Out[15]:
```
f_prime (generic function with 1 method)
```

In [18]:
```
f'(3.2)-f_prime(3.2)
```

Out[18]:
```
0.0
```

Das funktioniert für beliebige Funktionen.

In [19]:
```
function fact(n)
    if n<=1
        return 1.0
    end
    return n * fact(n-1)
end

fact'(3.2)
```

Out[19]:
```
13.520000000000003
```

Oder multivariate Funktionen (dafür benutzen wir die `gradient` Funktion des `Zygote` Pakets):

In [20]:
```
g(x,y)=x*y

gradient(g,3,7)
```

Out[20]:
```
(7, 3)
```

Hantieren mit Daten: `DataFrames` und `CSV/CSVFiles`

`DataFrames` stellt eine Datenstruktur für tabulare Daten zur Verfügung (Dokumentation).

Mit `CSV` und `CSVFiles` können tabulare Daten importiert und exportiert werden.

In [21]:
```
using DataFrames
using CSV
using CSVFiles
```

Ein DataFrame kann zum Beispiel so erstellt werden:

In [22]:
```
data=DataFrame(Name=["Hans", "Anna", "Klaus", "Petra"], Alter=[55,23,124,3],Geschlecht=["M","W","M","W"])
```

Out[22]:
4 rows × 3 columns

	Name	Alter	Geschlecht
	String	Int64	String
1	Hans	55	M
2	Anna	23	W
3	Klaus	124	M
4	Petra	3	W

Wir können weitere Zeilen mit `push!` hinzufügen

In [23]:
```
push!(data,["Goofy", 93, "M"])
```

Out[23]:
5 rows × 3 columns

	Name	Alter	Geschlecht
	String	Int64	String
1	Hans	55	M
2	Anna	23	W
3	Klaus	124	M
4	Petra	3	W
5	Goofy	93	M

Es sind viele Funktionen definiert um die Daten in der Tabelle zu verarbeiten (siehe Dokumentation). Z.B. können wir nur die Frauen aus der Liste heraussuchen:

In [24]:
```
filter(row->row.Geschlecht=="W", data)
```

Out[24]:
2 rows × 3 columns

	Name	Alter	Geschlecht
	String	Int64	String
1	Anna	23	W
2	Petra	3	W

Die Tabelle kann mit `CSV.write()` exportiert werden.

In [25]:
```
CSV.write("test.csv", data)
```

Out[25]:
```
"test.csv"
```

In [26]:
```
newdata = DataFrame(load("test.csv"))
```

Out[26]:
5 rows × 3 columns

	Name	Alter	Geschlecht
	String	Int64	String
1	Hans	55	M
2	Anna	23	W
3	Klaus	124	M
4	Petra	3	W
5	Goofy	93	M