

## InterCall - eine Mathematica-Schnittstelle zu externen Programmen

InterCall ist auf der Convex verfügbar und wird verwendet, um aus Mathematica über "mathlink" eine Verbindung zu externen Programmen, hauptsächlich NAG-Routinen, aber auch eigenen in Fortran oder C geschriebenen, herzustellen. Diese können in der Mathematica-Sitzung direkt gerufen werden. Die Ein- und Ausgabeparameter werden als Mathematica-Ausdrücke übergeben. So kann man die Ergebnisse mit allen Vorteilen, die Mathematica bietet, komfortabel weiterverarbeiten.

Installation :

Um InterCall zu nutzen, führt man :

```
/users/p0101/h0101alz/InterCall/UserInterCall.install
```

aus. Im Home-Verzeichnis findet man ein Mathematica-Programm InterCall.m, das in späteren Mathematica-Sitzungen zur Nutzung von InterCall gerufen wird. Beispiele sollen den Umgang mit InterCall erläutern.

Zu Beginn wird das Paket InterCall geladen

```
<<InterCall.m
```

```
Loading InterCall version 2.1.
```

```
Copyright (c) 1992-93 T. D. Robb.
```

Mit `?InterCall*` könnte man sich die neuen Kommandos auslisten lassen. Durch InterData wird der Zugriff auf die NAG-Numerikroutinen ermöglicht

```
<<InterData.m
```

InterCall stellt zu jeder NAG-Routine Kurzkommentare zur Verfügung. Mit `find` kann man nach Schlüsselwörtern oder Abkürzungen dieser suchen. Man erhält die Namen der Routinen, die das Gewünschte leisten!

```
find["system"&&"ordi"&&"diff"&&"equ"&&"
initial"&&Not["stiff"]&&"Rung"&&"
simpl"&&(NAG)"]
```

```
D02BAF (NAG)
```

```
System of ordinary differential
equations, initial value problem,
Runge-Kutta-Merson method, (simple
driver), over a range.
```

```
D02BBF (NAG)
```

```
System of ordinary differential
equations, initial value problem,
Runge-Kutta-Merson method, (simple
driver), over a range with inter-
mediate output.
```

```
D02BGF (NAG)
```

```
System of ordinary differential
equations, initial value problem,
Runge-Kutta-Merson method, (simple
driver), until a component of the
solution attains a given value.
```

```
D02BHF (NAG)
```

```
System of ordinary differential
equations, initial value problem,
Runge-Kutta-Merson method, (simple
driver), until a function of the
solution is zero.
```

**GetDefault** zeigt die voreingestellten Definitionen und Werte der entsprechenden Routine, hier von der Fortran-Subroutine

```
d02baf[x,xend,n,y,tol,fcn,w,ifail],
```

die es gestattet, ein Anfangswertproblem eines Systems gewöhnlicher Differentialgleichungen zu lösen (siehe oben). Die Parameter, die das Schlüsselwort "In" zugewiesen bekommen, sind beim Aufruf einzugeben (x,xend,y,fcn), während "Out" die Ergebnisparameter kennzeichnet. Die anderen Parameter sind vordefiniert, müssen beim Aufruf, wie in der letzten Zeile der folgenden Ausgabe angedeutet, nicht angegeben werden, es sei denn, man wünscht sie zu überschreiben. tol erhält beispielsweise den Wert 1.e-6, n den Wert, der sich aus der Dimension des Vektors y ergibt.

**GetDefault[d02baf]**

```
D02BAF[ (* TYPE=S *)
$X -> In, (* DATA=R *)
$XEND -> In, (* DATA=R *)
$N -> ROWS[$Y], (* DATA=I *)
$Y -> In :> Out, (* DATA=R[$N] *)
$TOL -> 10^(-6), (* DATA=R *)
$FCN -> In, (* DATA=S[R,
R[$N], R[$N]] *)
$W :> Null, (* DATA=R[$N, 7]
*)
$IFAIL -> -1 (* DATA=I *)
] (* CODE="LIBRARY" *)
d02baf[$X_, $XEND_, $Y_, $FCN] -> $Y
```

In Mathematica kann man nun die Funktion `fcn` definieren, die die rechten Seiten des Differentialgleichungssystems beschreibt.

### Hier das erste Beispiel!

*Flugbahn eines Projektils, das unter dem Winkel phi mit einer Geschwindigkeit v abgeschossen wird.*

Nutzer ohne Fortrankenntnisse können die Fortranbibliothek von NAG nutzen, ohne selbst in Fortran programmieren zu müssen.

```
mysub=Function[{x,y,f},Module[{f1,f2,f3,h,phi,v,
  grav=.032,drag=.02},{h,phi,v}=y;
  f1= Tan[phi];
  f2 = -grav/v^2;
  f3 = (f1*f2-drag/Cos[phi])*v;
  f={f1,f2,f3};(*Print["x=",x];*) ]];
```

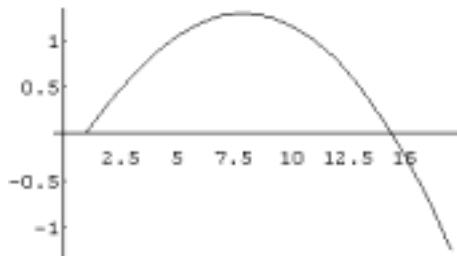
Mit dem folgenden Kommando wird d02baf bei festen Anfangswerten  $h=0$ ,  $\phi=\pi/5$ ,  $v=0.5$  mehrmals gerufen, wobei jeweils nur das Intervallende von 0 bis 8 in Schritten von .5 variiert, um eine Bahnkurve zu erhalten. Ausgegeben wird eine Tabelle von Wertetripeln für  $\{h,\phi,v\}$  am jeweiligen "xend".

```
points=Table[d02baf[0,x,{0,Pi/5,.5},mysub],
  {x,0,8,.5}];
```

```
InterCall::opened: Opened connection
  to host joker
InterCall::import: Importing:
  {D02BAF}
InterCall::linked:
  Using remote driver version 2.1
  on host joker
```

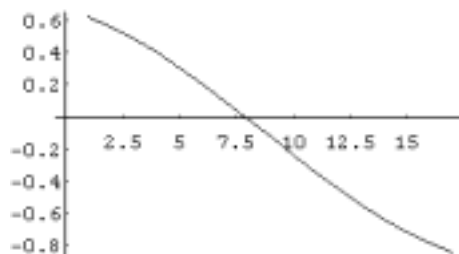
Mit einem einzigen Schritt kann man etwa die ersten, zweiten, dritten oder alle Werte, also die Höhen, Winkel, ... des Projektils, äquidistant darstellen.

```
h=ListPlot[Transpose[points][[1]],
  PlotJoined->True]
```



-Graphics-

```
phi=ListPlot[Transpose[points][[2]],
  PlotJoined->True]
```



-Graphics-

### Ein zweites Beispiel!

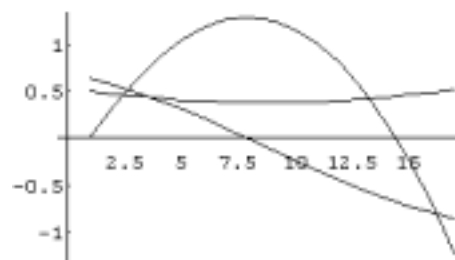
Gegeben sei eine selbst geschriebene Fortranroutine, die nach dem Newtonverfahren, ausgehend von einem Startwert, eine Nullstelle einer zu definierenden Funktion *func* ermittelt. Hier ist das Listing der Fortranroutine

```
v=ListPlot[Transpose[points][[3]],
  PlotJoined->True]
```



-Graphics-

```
Show[h,phi,v]
```



-Graphics-

Man kann auch durch einen einmaligen Aufruf der Routine die Anfangswerte erklären, diese dann durch **SetDefault** festschreiben und vereinfacht fortfahren, wie etwa :

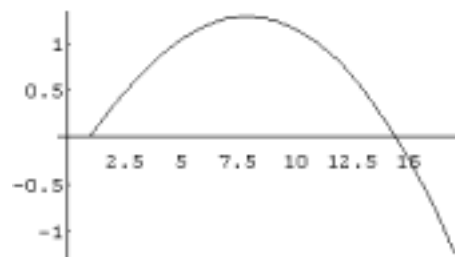
```
d02baf[0,0,{0,Pi/5,.5},mysub]
```

```
{0, 0.628318530717959, 0.5}
```

```
SetDefault[d02baf,$X->$X,$Y->$Y:>Out,
  $FCN->$FCN];
```

```
points=Table[d02baf[x],{x,0,8,.5}];
```

```
ListPlot[First /@ points,
  PlotJoined->True];points=.
```



-Graphics-

mynewt.f, die sich zusammen mit dem Objektmodul im vom Home-Verzeichnis aus gesehenen Knoten fx/InterCall befindet.

### !!fx/InterCall/mynewt.f

```
C % f77 -c mynewt.f
C mynewt.f: NEWTON
      subroutine NEWTON(X,FUNC,GRAD,EPS,MAXITS)
      integer MAXITS,ITER
      real*8 X,FUNC,GRAD,EPS,XGUESS
      external FUNC,GRAD
      ITER = 0
10     XGUESS = X
      ITER = ITER + 1
      X = XGUESS - FUNC(XGUESS)/GRAD(XGUESS)
      if ( abs(X-XGUESS).gt.EPS .and. ITER.lt.MAXITS ) goto 10
      if (ITER.ge.MAXITS) print *,'NOT CONVERGING'
      end
```

<<InterCall.m

<<InterData.m

Loading InterCall version 2.1.

Copyright (c) 1992-93 T. D. Robb.

Get::noopen: Cannot open idata\_LINPACKD.m.

**AddDefault** erklärt die Nutzung der Parameter, die groß zu schreiben sind und ein „\$“ am Anfang erhalten. Die Stärken von Mathematica sind hier bei der Bildung des Gradienten besonders ersichtlich: Ein einziges Mathematica-Kommando löst das Problem!

```
AddDefault[NEWTON[$X->In:>Out,$FUNC->In,
  $GRAD->Derivative[1][$FUNC],$EPS->10^-9,
  $MAXITS->20],S[R,RF[R],RF[R],R,I],
  "fx/InterCall/mynewt.o"]
```

NEWTON

Zur eigenen Sicherheit:

**GetDefault[newton]**

```
NEWTON[ (* TYPE=S *)
  $X -> In :> Out, (* DATA=R *)
  $FUNC -> In, (* DATA=RF[R] *)
  $GRAD -> Derivative[1][$FUNC],
  (* DATA=RF[R] *)
  $EPS -> 10^(-9), (* DATA=R *)
  $MAXITS -> 20 (* DATA=I *)
] (* CODE="fx/InterCall/mynewt.o"
*)
newton[$X_, $FUNC_] -> $X
```

**?newton**

```
newton[$X_, $FUNC_] -> $X
```

Alles ist richtig konstruiert. Man kann nun **newton** nutzen. Zunächst mit einer im Sprachgebrauch von Mathematica üblichen *reinen* Funktion Sin[#] -#&

```
newton[2.,2*Sin[#]-#&]
```

InterCall::opened: Opened connection to host joker

InterCall::import: Importing: {NEWTON}

InterCall::linked:

Using remote driver version 2.1 on host joker

1.89549426703398

In der Nähe des Startwertes 2. liegt die Nullstelle 1,89... . Probe :

```
2 Sin[%]-%
-15
1.77636 10
```

Andere Funktion:

```
newton[4.,BesselJ[1,#]&]
```

3.83170597020751

```
newton[0.2,BesselJ[1,#]&]
```

0

```
BesselJ[1,3.83170597020751]
```

```
-15
1.05073 10
```

```
BesselJ[1,0]
```

0

Eine Beschreibung von InterCall ist über die *mathsource* verfügbar. Weitere Hinweise kann man beim Autor erhalten.

Hans-Joachim Spitzer