CvO Universität Oldenburg
Institut für Mathematik
Prof. Dr. Hannes Uecker

8. Mai 2009

Über Software zum Plotten von Kurven und Vektorfeldern
(und zum numerischen Lösen von gDGL u.ä.)

Hier ein paar Tipps zu software, die eventuell bei Problemen der Analysis 2 hilfreich ist. Diese spiegeln natürlich meinen persönlichen Geschmack wieder.

Vorbemerkungen. a) Grundsätzlich gilt: keine Software (oder Taschenrechner) kann Ihnen das eigene Nachdenken abnehmen. Um eine Software sinnvoll einzusetzen, müssen Sie das mathematische Problem verstehen, und möglichst schon eine grobe Vorstellung der Lösung haben. Im (Grund)studium sollten Sie Software nur gelegentlich zur Visualisierung und zur Kontrolle verwenden, oder zur elektronischen Dokumentation (Proseminar etc). Vom Gebrauch von Software als Gewohnheitstool für "Kurvendiskussion" (reeller Funktionen) und sonstigen Analysis 1 Stoff rate ich dringend ab.
b) Für die Analysis 2 gilt im Prinzip das gleiche, allerdings ist hier die Visualierung meist ungleich nützlicher. Darum seien hier vier Programme zum "Plotten" (und viel mehr) vorgestellt. Dabei schränke ich mich auf "freie" Software ein, die nicht weniger leistungsfähig und sogar flexibler als die kommerzielle Konkurrenz ist, und zwar `gnuplot` und `octave`, sowie auf das einfache Java–Applet `phaseplane` zum Plotten von Phasenporträts (Vektorfeld+Integralkurven in 2D), und schließlich auf das sehr mächtige Programm `xppaut`.
c) Nicht betrachtet werden "Computeralgebrasysteme" (Maple, Mathematica bzw. die freien Konkurrenten). Diese können auch (fast) alles, was unten vorgestellt wird, und viel mehr, schiessen aber für unsere Zwecke über das Ziel hinaus[1].
d) Untige Bemerkungen beziehen sich auf `gnuplot`, `octave` und `xppaut` unter einem generischen `linux`, wozu auch an dieser Stelle jedem geraten sei. Die Programme laufen auch unter Windows und Mac. `phaseplane` ist unabhängig vom OS und braucht nur einen Java fähigen Browser.

1 gnuplot

`gnuplot`, `http://www.gnuplot.info/` [1], ist ein Kommandozeilen–orientiertes (kein Menu-rumgeklicke nötig!) Plot-Program, das Ihnen (fast) alles plottet, was Sie jemals brauchen. Unter [1] finden sich auch diverse Hilfen, Demos, Tutorien, auch was die diversen Ausgabemöglichkeiten (eps, pdf, jpg, png usw) angeht. Nachfolgende Beispiele sollen nur als Einstieg dienen. In die Kommandozeile einzugebende Befehle sind dabei als `truetype` gesetzt.

1. `plot [-pi:pi] exp(cos(x))` 1D Funktion, erklärt sich von selbst! Mehrere Funktionen in einen plot erhält einfach durch Kommas. Beispiel `plot [0:pi] exp(cos(x)), 2*x lw 3` wobei `lw 3` für linewidth 3 steht (und also einen dickeren Stift einstellt). Man kann alle Befehle (eindeutig!) abkürzen, z.B. `p` für `plot` tippen, aber das kann man natürlich auch übertreiben.

2. `splot [-1:1] [-pi:pi] x**2+sin(y)` plot einer Funktion $f : \mathbb{R}^2 \to \mathbb{R}$ (später in der Vorlesung), splot wie surface plot; dazu können auch contours (Höhenlinien) sowie alles mögliche ge-

[1] Sie können z.B. "viele" Differentialgleichungen symbolisch (d.h.in geschlossenen Formeln) lösen, und natürlich symbolisch differenzieren und integrieren

geben werden. Die "Standarddarstellung" ist etwas "arm" (Gitterplot), aber man kann alles beliebig aufpeppen, Beispiel: `set cont surf; set isosamp 50,50; set pm3d; unset surf; splot [-1:1] [-pi:pi] x**2+sin(y)` (Dies kann man so in die Kommandozeile eingeben, oder alternativ statt jedem ; return drücken. Um Optionen wieder auszuschalten setzt man einfach z.B. `unset cont`; vorherige Eingaben bekommt man mit den Pfeiltasten zurück und kann sie dann modifizieren.)

3. Parametrische plots (allgemeine Kurven im $\mathbb{R}^2$ oder Flächen im $\mathbb{R}^3$). Zunächst `set para` liefert die Mitteilung `dummy variable is t for curves, u/v for surfaces`. Kreis plotten: `plot [0:2*pi] cos(t),sin(t)`. Kardioide plotten erklärt sich dann auch von selbst:
`plot [0:2*pi] 2*cos(t)-cos(2*t),2*sin(t)-sin(2*t), 1*cos(t), 1*sin(t), 2+cos(t), sin(t)`
(insgesamt plotten wir die Kardioide, den inneren Kreis, und den äußeren Kreis, deshalb 3*2=6 Koordinaten; man experimentiere mit Strichstärken `lw z` und linetypes `w l m`, $z > 0, m \in \mathbb{N}$). Beipiel für parametrisierte Fläche (siehe später in VL): `set view 60,120; set urange [-3:3]; set vrange [-3:3]; set title 'some example'; splot u**2-v**2,2*u*v,v`

4. `gnuplot` ist eine vollwertige Programmiersprache! D.h. man kann sich z.B. Variablen und eigene Funktionen definieren, und diese dann plotten. Man kann Kommandos in einem beliebigen Text-file speichern [2], z.B.`epi.gp`, und dieses dann mit `load 'epi.gp'` einlesen. Beispiel:

```
epix(t)=(br+sr)*cos(t)-a*cos((br+sr)/sr*t); % x-koord einer Epizykl. mit R=br, r=sr, a=a
epiy(t)=(br+sr)*sin(t)-a*sin((br+sr)/sr*t); % y-koord
br=5.1; sr=1; a=1.5; plot epix(t),epiy(t);  % ein Beipiel
```

   Wenn einem die Ausgabe noch nicht gefällt kann man jetzt einfach in der Kommandozeile weitermachen, z.B. `set samples 100; plot epix(t),epiy(t);`

Damit haben wir ganz leicht an der Oberfläche von `gnuplot` gekratzt. `gnuplot` ist ebenfalls extrem nützlich zum Plotten (beliebig komplizierter) Daten aus Dateien. `gnuplot` kann auch Vektorfelder plotten, aber dies gibt uns Gelegenheit ein weiteres Programm vorzustellen, das alles obige natürlich auch kann (und `gnuplot` als plot-machine verwendet), und viel mehr, u.a. eben Vektorfelder plotten sowie gewöhnliche Differentialgleichungen numerisch lösen, und all dies sehr benutzerfreundlich.

2 `octave`

`octave`, `www.gnu.org/software/octave/` [2], ist eine Programmiersprache (oder Kommandozeilen–orientierte Programmierumgebung) für numerische Berechnungen (oder einfach nur zum Plotten). D.h. es werden "high-level" Routinen zur Verfügung gestellt, mit denen typische Berechnungen wie Lösen eines LGS o.ä. zu einem einfachen Befehl werden, z.B. `x=A\b` zum Lösen von $Ax = b$ mit $A \in \mathbb{C}^{n\times n}$ und $b \in \mathbb{C}^n$. All dies ist vollständig online dokumentiert, z.B. unter [2].

Hier konzentrieren wir uns auf das Plotten von Vektorfeldern. Dies kann einfach von der Kommandozeile aus erledigt werden, aber zwecks Kommentierung schreibe ich die Befehle in eine Datei `vf1.m`, die dann via `vf1` von `octave` aus aufgerufen und sequentiell abgearbeitet wird[3]. Beispiel (Kommentare beginnen %):

[2] hierzu verwende man den einzig wahren Editor `emacs` oder eventuelle Abkömmlinge, `www.gnu.org/software/emacs/`

[3] dazu muss `vf1.m` natürlich im Suchpfad sein, z.B. im aktuellen Verzeichnis. `octave` ist (fast) vollständig kompatibel zu `matlab` – oder umgekehrt, was die Dateiendung .m erklärt. Wer `matlab` hat, kann alles was folgt auch mit `matlab` machen.

```
% Beipiel für plot eines VF
x=-2:0.2:2; % in octave basiert alles auf Vektoren (arrays), deshalb wird zunächst
% ein solcher angelegt (der hier das Intervall [-2,2] mit Schrittweite 0.2 diskretisiert)
y=-1:0.1:1;  % analog für y
[X,Y]=meshgrid(x,y); % damit ist dann [-2,2]x[-1:1] diskretisiert
quiver(X,Y,Y,-X); % das wär's: 1. und 2. Argument sind das Gitter auf dem das
%Vektorfeld gezeichnet wird, 3. und 4. Argument sind 1. und 2. Komponente des VF
```

Um Infos zu Kommandos zu erhalten ist neben der Doku das Kommando (Beispiel) `help quiver` nützlich. Nachdem die Matrizen `X,Y` definiert sind, können wir auch andere Vektorfelder von der Kommandozeile plotten, z.B. `quiver(X,Y,Y,-sin(pi*X);` oder `quiver(X,Y,Y,-X+X.^3);` Letzteres soll auf eine (anfangs verwirrende) Besonderheit hinweisen: in `octave` gibt es neben der normalen Arithmetik `a*b`$= ab$ oder `a/b`$= ab^{-1}$, wobei a, b beliebige (kompatible) Matrizen sein können, die (fast wichtigere) "array-smart" Arithmetik `a.*b` und `a./b`, die komponentenweise agiert, d.h. `a.*b` ist die Matrix $(a_{ij}b_{ij})$. D.h `X.^3` ist etwas völlig anderes als `X^3`. Dies mag zunächst umständlich wirken, ist aber ungemein praktisch und erspart (fast) alle "low-level" Schleifen. Die Standardfunktionen ($\sin, \cos, \exp$ usw.) sind alle "array-smart", d.h. $\sin(X) = (\sin(x_{ij})_{ij})$. Bei Definition eigener Funktionen verwende man als Standard ebenfalls die "array-smart" Variante, selbst wenn man glaubt, nur Skalare zu benötigen: `*` statt `.*` verwende man nur, wenn man explizit das Matrixprodukt meint. Man experimentiere hiermit, und denke bei Fehlern stets zuerst an Verwechslung von z.B. $*$ und $.*$!

Als Nachtrag zu §1 plotten wir noch Raumkurven, z.B. via `t=0:0.1:5; plot3(cos(t),sin(t),t)`. Zum Plotten von Vektorfeldern $f : \mathbb{R}^3 \to \mathbb{R}^3$ verwendet man `quiver3`, Beispiel: `x=0:0.1:1; y=x; z=x; [X,Y,Z]=meshgrid(x,y,z); quiver3(X,Y,Z,X,Y,Z.^2);`

3 Plotten von Phasenporträts

Wir wenden uns der numerischen Lösung von DGl[4] und damit Integralkurven zu, oder allgemeiner dem Plotten von Phasenporträts. Grundsätzlich ist empfehlenswert, solche tools einmal selbst zu programmieren (z.B. in `octave`, was auch hierfür sehr viele "high-level" Kommandos bereits bereitstellt), aber das geht über diese Vorlesung hinaus.

Ein sehr simples JAVA–applet, das sich selbst erklärt, ist `phaseplane`, siehe `http://www.math.rutgers.edu/courses/ODE/sherod/phase-local.html` [5]. Applets dieser Art gibt es jede Menge, zum Einstieg sind sie völlig ausreichend, aber sie stossen auch schnell an ihre Grenzen. Darum sei noch auf das Programm `xppaut` hingewiesen, das *alle* Standardfragen (und vieles mehr) behandeln kann, und zwar in beliebigen Raumdimensionen. Das folgende Tutorial ist in Englisch, sorry! In diesem werden ausserdem einige Konzepte verwendet ("invariant manifolds", "Hamiltonian"), die in dieser Vorlesung nicht behandelt werden. Da sie später nützlich sein könnten, hab ich sie drin gelassen. ODE steht dabei übrigens für ordinary differential equation. Zur (nicht ganz trivialen) Windows Installation sei noch auf `www.staff.uni-oldenburg.de/hannes.uecker/soft.html` verwiesen.

[4] **Vorsicht:** Numerik von DGl ist ein weites Feld, das in Spezialvorlesungen behandelt wird. Hier geht es nicht um die Numerik, sondern nur um Beipiele zu 2D gDGL, d.h. es werden nur "black-box" Programme angewandt

[5] siehe auch pplane.tar unter Stud.IP für den offline Betrieb

3.1 First steps in xppaut

xppaut, www.math.pitt.edu/∼bard/xpp/xpp.htm, is a very powerful program to study ODEs. It is freely available, has extensive documentation and tutorials, both online and in print, and only requires a minimal installation effort. Here we give a short tutorial of xppaut, under linux, but under Windows it is just the same.

We consider the 2D ODE system

$$\partial_t x = y \quad \text{and} \quad \partial_t y = -cy + x - x^3. \tag{1}$$

with parameter $c \in \mathbb{R}$. This system belongs to the scalar ODE $\partial_t^2 x = -c\partial_t x + x - x^3$ which is a Newtonian ODE with damping for $c > 0$. For $c = 0$ we have a Hamiltonian system, and the term $f(x) = x - x^3$ is similar to the Taylor expansion of $\sin(x) = x - x^3/3! + \ldots$. The Newtonian structure will be helpful for the analytical understanding, but first we are interested in the numerical investigations.

To start we write an "ODE file" example.ode for xppaut in the form

```
x'=y
y'=-c*y+x-x**3
param c=0
@ background=White          # printer-friendly background
@ xlo=-2,xhi=2,ylo=-2,yhi=2 # window size
@ xp=x,yp=y       0          # plotting variables
done
```

The file essentially explains itself. The line @ background=White sets a printer friendly white background. We start xppaut by typing xppaut example.ode and obtain the X–window in Fig.1a).

The large window shows the x-y plane, on top there are a number of buttons which open further windows, and on the left there are some action buttons. We now want to carry out our standard program of first finding fixed points, their stability and their invariant manifolds. Alternatively, one can imediately plot the director field/flow phase portrait to get an overall impression, but we postpone this for a minute to take an approach closer to analysis. We do all this for $c = 0$ and encourage the reader to repeat the steps for different parameter values of c, which can be changed by clicking Param at the top.

In fact, for (1) most of our numerical study can be carried out analytically, but as already said the purpose of this section is to explain xppaut. In the following the symbol -> means to choose an option from a submenu.

A useful tool to find fixed points is to let xppaut first show the *nullclines*. For a C^1 vectorfield

$$f = (f_1, \ldots, f_d) : \mathbb{R}^d \to \mathbb{R}^d$$

there are d nullclines $N_1, \ldots, N_d$, which are generically $d-1$ dimensional submanifolds of $\mathbb{R}^d$ and defined by

$$N_i = \{x \in \mathbb{R}^d : f_i(x) = 0\}.$$

In $d = 2$ the nullcline N_1 (resp. N_2) describes the points where f is vertical (resp. horizontal). To find fixed points we need intersections of all nullclines.

xppaut calculates nullclines after clicking Nullcline. We get $N_1 = \{(x, 0) : x \in \mathbb{R}\}$ and $N_2 = \{(x, y) : x = 0 \text{ or } x = \pm 1, y \in \mathbb{R}\}$. Clearly we have three intersections. To let xppaut find the fixed points we click Sing points-> Mouse and are asked for an initial guess. Clicking near $(x, y) = (0, 0)$

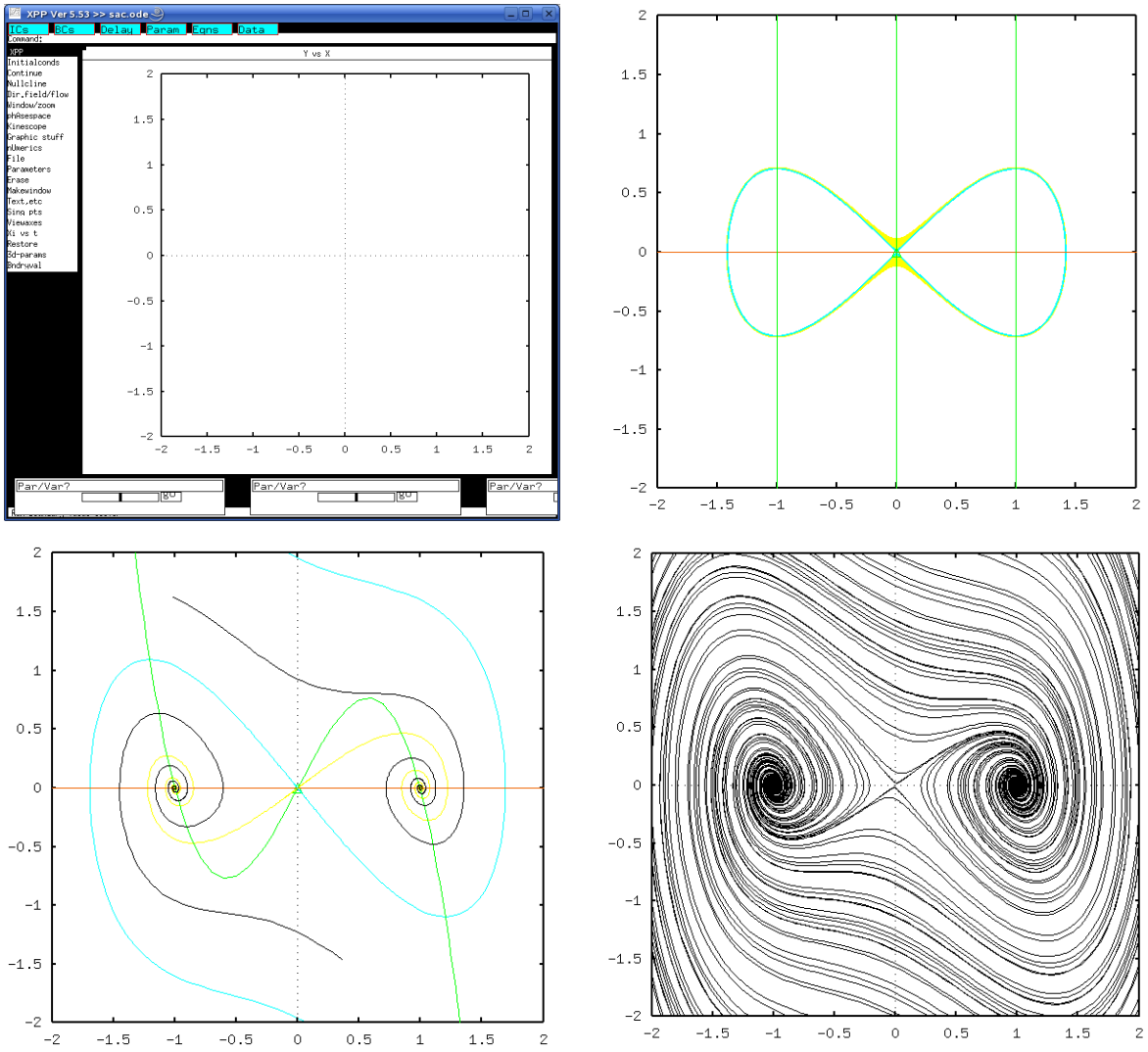

Abbildung 1: `xppaut` screenshots.

we get (not surprisingly) the fixed point $(0,0)$. Moreover, `xppaut` shows the eigenvalues $\lambda_{\pm} = \pm 1$, and draws a triangle around $(0,0)$ to indicate that it is a saddle. Next, `xppaut` suggests to draw the invariant sets, i.e., the stable and unstable manifold of $(0,0)$, and after confirmation we get the picture in Fig.1b). Here we need to point out an almost unavoidable problem of all numerical simulations. As explained below, it turns out that the unstable and stable manifolds of $(0,0)$ are identical, i.e., the unstable manifold of $(0,0)$ leaving $(0,0)$ along $(1,1)$ comes back to $(0,0)$ along $(-1,1)$ as stable manifold. Therefore, $W^u = W^s$ gives a figure eight. However, in the numerical calculation of W^u numerical errors have the effect that $W^u = W^s$ does not come back to $(0,0)$. It comes close to $(0,0)$, but since $(0,0)$ is unstable, the numerical approximation W^s_n leaves $(0,0)$ again along the unstable direction $(-1,-1)$, and alltogether W^u_n turns around $(0,0)$ forever. There are ways around this unwanted behaviour, but these need specialized (symplectic) integrators. Here, a remedy is to abort the calculation of W^u_n by hitting `esc` (two times). This causes `xppaut` to start the calculation of W^s_n, and since a similar numerical problem occurs we again hit `esc` two times. This gives the picture in Fig.1b).

At the two other fixed points $(\pm 1, 0)$, `xppaut` finds the eigenvalues $\pm i\sqrt{2}$; thus these are centers, and there is no point for `xppaut` to calculate W^c since they are 2–dimensional.

Now changing the param c to, e.g., $c = 0.5$, we find that $(0, 0)$ remains a saddle, while $(\pm 1, 0)$ turn into sinks. As a consequence, $W^u((0, 0))$ now connects $(0, 0)$ with $(\pm 1, 0)$, and $W^s((0, 0))$ comes from infinity, and both are correctly approximated by `xppaut`. See Fig.1c) for the nullclines, fixed points and invariant manifolds, and for two orbits corresponding to specific initial conditions. The latter are obtained by clicking `Initialconds->Mice` and clicking into the plane for choosing the initial conditions. Finally Fig.1d) shows a flow portrait, obtained by clicking `Dir.field/flow -> Flow` and hitting `return`.