Institut für Parallele und Verteilte Systeme
Abteilung Bildverstehen
Universität Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Studienarbeit Nr. 2181

Design und Implementierung einer grafischen Benutzerschnittstelle für AnT 4.669

Andreas Wild

Studiengang:	Informatik
Prüfer:	PD Dr. rer. nat. Michael Schanz
Betreuer:	Dr. rer. nat. Viktor Avrutin
begonnen am:	1. Mai 2008
beendet am:	1. November 2008
CR-Klassifikation:	H.5.2, H.1.2

Inhaltsverzeichnis

Abbildungsverzeichnis

Verzeichnis der Listings

Abkürzungsverzeichnis

GTK+
: GIMP Toolkit

GUI
: Grafische Benutzerschnittstelle

Teil I

Überarbeitung der grafischen Benutzerschnittstelle von AnT 4.669

Kapitel 1

Einleitung

AnT 4.669 ist ein Softwarepaket, das zur Simulation und Untersuchung von dynamischen Systemen durch die Nichtlineare Dynamische Systeme-Gruppe an der Universität Stuttgart entwickelt wurde[1]. Es unterstützt die gängigsten Klassen von dynamischen Systemen, sowohl diskrete als auch kontinuierliche, sowie eine Vielzahl von Analysemethoden, wie zum Beispiel die Untersuchung auf Periodizität oder die Bestimmung von Lyapunov-Exponenten. Die berechneten Daten werden in Textdateien geschrieben, die danach weiter verarbeitet oder visualisiert werden können – beispielsweise mit Hilfe von gnuplot[2]. Um ein System mit AnT 4.669 zu untersuchen, muss man es üblicherweise in der Sprache C implementieren und als eine dynamische Bibliothek kompilieren, die dann zur Laufzeit von AnT 4.669 verwendet werden kann.

Die Initialisierung des Systems durch die Auswahl von Systemparametern, Anfangs- und Laufbedingungen und durchzuführenden Analysemethoden erfolgt durch Konfigurationsdateien. Um insbesondere bei der Neuerstellung von solchen Konfigurationen einen Überblick über die vorhandenen Möglichkeiten und Parameter zu bekommen, empfiehlt es sich, die mitgelieferte Grafische Benutzerschnittstelle (GUI), die speziell für die Bearbeitung der Konfigurationsdateien entwickelt wurde, zu verwenden. Gegenüber der manuellen Bearbeitung der Konfiguration im Texteditor stellt diese GUI bereits einen großen Schritt dar, jedoch ist die Benutzerführung inzwischen nicht mehr zeitgemäß. Das Ziel dieser Arbeit ist nun, die Benutzerfreundlichkeit der GUI an heutige Standards anzugleichen.

[1] `http://www.ant4669.de`
[2] `http://www.gnuplot.info`

Kapitel 2

Planung

2.1 Ausgangssituation

AnT 4.669 stellt eine große Auswahl an Möglichkeiten zur Untersuchung von dynamischen Systemen zur Verfügung. Dadurch ist die Komplexität der Konfigurationsdateien entsprechend hoch. Diese Komplexität stellt gewisse Anforderungen an die GUI:

Aufgrund der großen Anzahl an verfügbaren Optionen sind Konfigurationsdateien für AnT 4.669 hierarchisch aufgebaut (siehe Listing 2.1). Dieser Aufbau wird in der GUI genutzt, indem nur die Teile der Konfiguration angezeigt werden, die sich auf dem aktiven Konfigurationspfad[1] befinden, denn eine Anzeige aller verfügbaren Optionen gleichzeitig ist nicht möglich.

Zusätzlich sind viele der Konfigurationsknoten voneinander abhängig, beispielsweise aktiviert die Auswahl der Systemklasse „map“ eine Reihe von spezifischen Methoden, die für diese Klasse relevant sind. Das bedeutet für die GUI, dass die Konfigurationsfenster nicht fest einprogrammiert werden können, sondern dynamisch generiert werden müssen. Die Gesamtheit des AnT 4.669-Konfigurationsbaumes wird ihrerseits in einer Meta-Konfigurationsdatei festgehalten, die Änderungen an der GUI ohne neukompilieren des Quellcodes ermöglicht.

Wird eine Konfigurationsdatei in der GUI geöffnet, so sind zunächst nur die Konfigurationsknoten der obersten Ebene sichtbar. Nach Auswahl eines solchen Knotens öffnet sich ein neues Fenster, in dem die zugehörigen Kindknoten sichtbar sind. Diese Kindknoten können nun entweder nur direkt veränderbare Optionen beinhalten oder ihrerseits auch wieder Kindknoten enthalten. In letzterem Fall öffnet sich nach der Auswahl wieder ein neues Fenster. Wegen der schon angesprochenen Abhängigkeiten der Optionen untereinander darf dabei nur das jeweils zuletzt geöffnete Fenster aktiv bleiben, um mögliche Inkonsistenzen in der Konfiguration zu vermeiden.

[1] Der Pfad durch den Konfigurationsbaum, der durch die Abfolge der Knoten von der Wurzel bis zum aktuellen Konfigurationsknoten gegeben ist.

Listing 2.1 Ausschnitt aus einer Konfigurationsdatei für AnT 4.669

```
dynamical_system = {
  type = map,
  name = "logistic",
  parameter_space_dimension = 1,
  parameters = {
    parameter[0] = {
      value = 0,
      name = "r"
    }
  },
  state_space_dimension = 1,
  initial_state = (0.1),
  reset_initial_states_from_orbit = false,
  number_of_iterations = 10000,
  s[0] = {
    name = "x",
    equation_of_motion = "r*x*(1-x)"
  }
},
scan = {
  type = nested_items,
  mode = 1,
  item[0] = {
    type = real_linear,
    points = 1000,
    min = 0,
    max = 4,
    object = "r"
  }
},
...
```

2.2 Problemanalyse

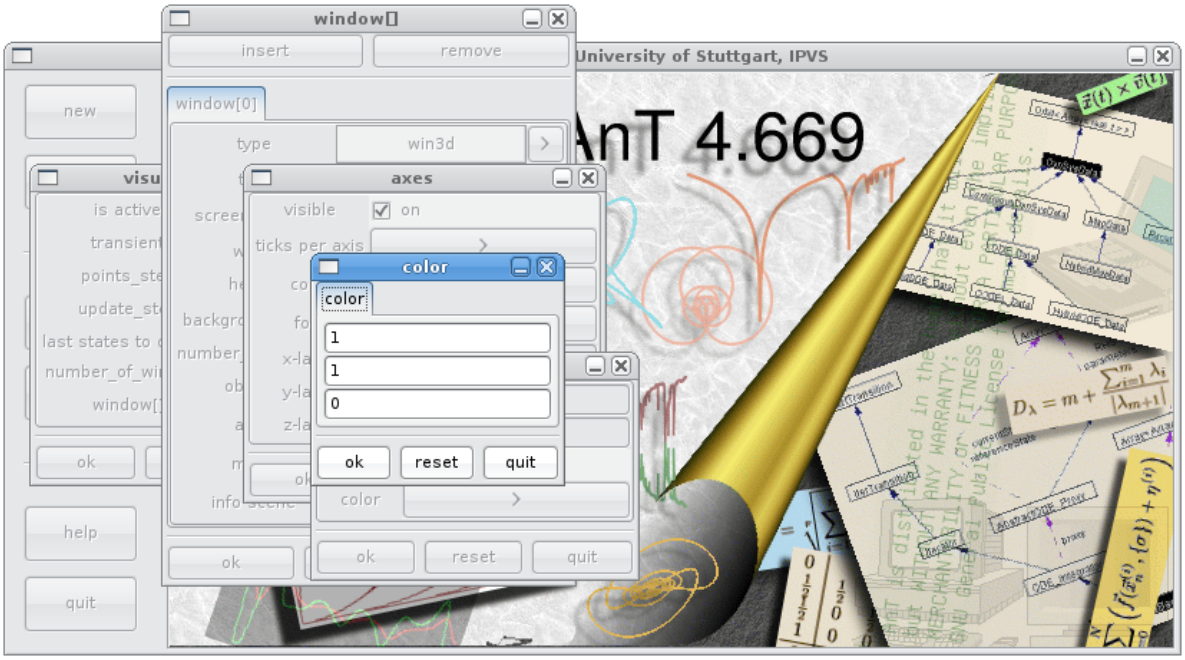

Abbildung 2.1: Mehrfenster-Lösung

In der aktuellen Version der GUI werden diese neu geöffneten Fenster eigenständig erzeugt und angezeigt (Mehrfensterlösung), was sehr schnell zu einer großen Anzahl an Fenstern auf dem Bildschirm führt. Darunter leidet zum einen die Orientierung des Nutzers im Konfigurationsbaum und zum anderen die Übersichtlichkeit bei der Bedienung von anderen, parallel laufenden Anwendungen. Abbildung 2.1 zeigt eine Beispielsitzung mit dieser Version der GUI.

2.3 Lösungsansätze

Im Rahmen dieser Arbeit wurden für die vorliegende Problematik zwei verschiedene Lösungsansätze diskutiert.

2.3.1 Reine Einfensterlösung

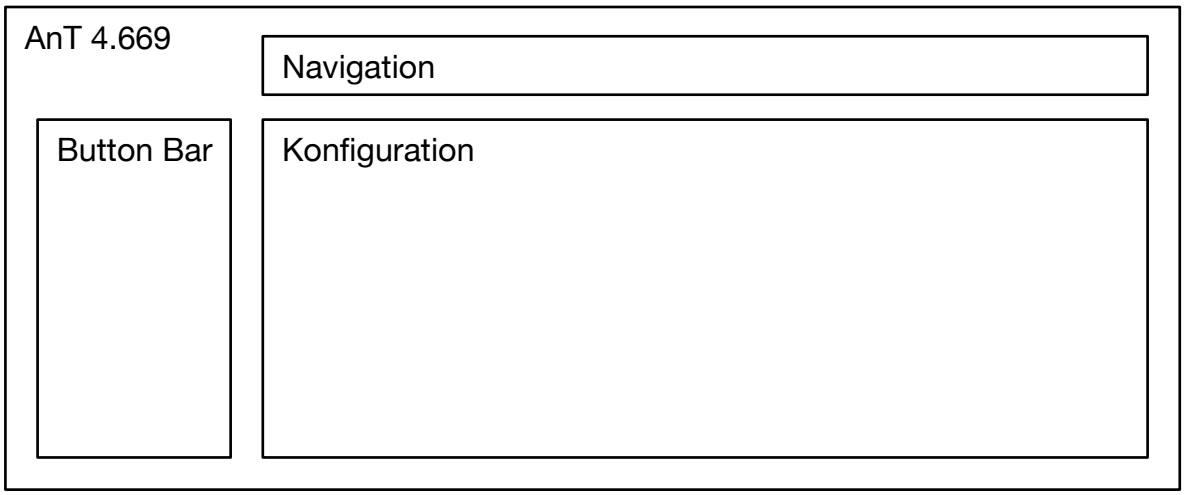

Abbildung 2.2: Reine Einfensterlösung

Ein erstes Konzept sah vor, die entstehenden Unterfenster in das Hauptfenster einzubetten und so zu überlagern, dass jeweils nur das obere sichtbar ist (Einfensterlösung). Dabei wird jedoch zwar die augenscheinliche Komplexität der Konfiguration verborgen, aber die Orientierung des Nutzers ist damit sogar schlechter als bei der Mehrfensterlösung, da es dann gar nicht mehr möglich ist, den bisherigen Pfad durch den Konfigurationsbaum (aktiver Konfigurationspfad) nachzuvollziehen. Eine Möglichkeit, dieses Problem anzugehen wäre, eine Navigationsleiste einzublenden, in der der aktive Konfigurationspfad sichtbar ist. Das erfordert jedoch einerseits aufwändige strukturelle Änderungen in der Erzeugung der Fenster und bietet andererseits nicht die Möglichkeit, den Inhalt vorheriger Konfigurationsfenster zu sehen.

2.3.2 Einfensterlösung mit Spaltenansicht

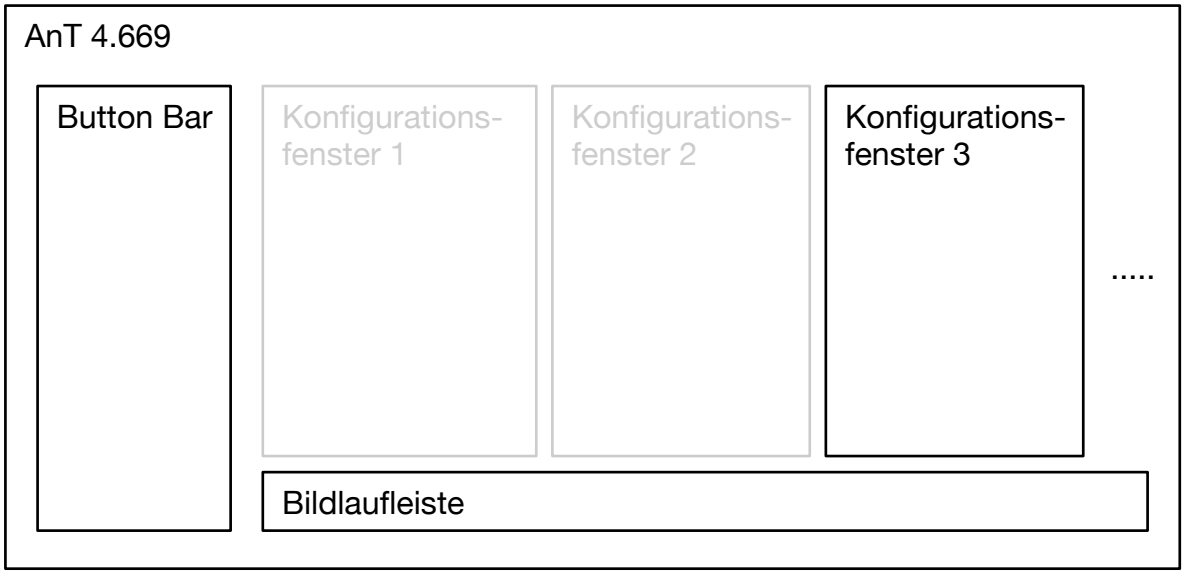

Abbildung 2.3: Einfensterlösung mit Spaltenansicht - inaktive Konfigurationsfenster grau dargestellt

Um den Schwächen der reinen Einfensterlösung entgegenzuwirken, wurde das Konzept erweitert: Die Unterfenster werden weiterhin in das Hauptfenster eingebettet, aber anstatt sie zu überlagern, werden sie in einer Reihe nebeneinander angeordnet (Spaltenansicht), so dass das jeweils zuletzt geöffnete Fenster ganz rechts zu sehen ist. Bei dieser Ansicht ist nun sowohl der aktive Konfigurationspfad als auch der Inhalt vorheriger Konfigurationsfenster dauerhaft sichtbar, was weder bei der Mehrfensterlösung noch bei der ersten Version der Einfensterlösung möglich war.

Kapitel 3

Entwicklung

3.1 Entwicklung des GtkColumnView-Widgets

3.1.1 Motivation

Die AnT 4.669-GUI wurde unter Verwendung des GIMP Toolkit (GTK+)[1] entwickelt. GTK+ ist eine plattformübergreifende OpenSource-Bibliothek zur Erstellung von GUIs und ist für UNIX, Linux, Windows und Mac OS X erhältlich. Fenster und Steuerelemente werden in GTK+ durch so genannte „Widgets“ repräsentiert. Im Rahmen dieser Arbeit wurde zur Darstellung der Konfigurationsfenster in einer Spaltenansicht das `GtkColumnView`-Widget entwickelt, um die gewünschte zusätzliche Funktionalität zu kapseln.

3.1.2 Anforderungen

Aufgrund der potenziell großen Tiefe der Konfigurationspfade ist es üblicherweise nicht möglich, alle Konfigurationsfenster gleichzeitig nebeneinander sichtbar zu halten. Eine der Aufgaben des `GtkColumnView`-Widgets besteht also darin, horizontale Bildlaufleisten zur Verfügung zu stellen, sobald die Breite aller sichtbaren Konfigurationsfenster aufsummiert größer als die Breite des Hauptfensters ist. Zusätzlich muss beim Öffnen und Schließen dieser Fenster sichergestellt sein, dass das jeweils aktiv werdende Fenster vollständig sichtbar ist.

Bei der Mehrfensterlösung richtete sich die Größe der erzeugten Fenster nach ihrem Inhalt. In der Spaltenansicht haben nun aber alle Fenster die gleiche Höhe. Dadurch können sich Schwierigkeiten ergeben, sobald der Inhalt des Fensters höher ist als die diese verfügbare, maximale Höhe. Um dieses Problem zu lösen, sollte einerseits die Größe des `GtkColumnView`-Widgets so gewählt werden, dass die größten Konfigurationsfenster vollständig dargestellt werden können. Andererseits muss aber auch die Möglichkeit vorgesehen werden, vertikale Bildlaufleisten einzublenden, da beispielsweise bei sehr komplexen Systemen lange

[1] `http://www.gtk.org`

Parameterlisten entstehen können, deren Größe erst zur Laufzeit bekannt ist. Außerdem dürfen Benutzer einerseits das Hauptfenster verkleinern und können andererseits betriebssystemseitig Steuerelemente und insbesondere Text größer darstellen lassen, was ebenfalls zu Platzmangel führen kann.

Diese Bildlaufleisten sollten idealerweise nur den Inhalt der Konfigurationsfenster scrollen und nicht die zugehörigen Buttonleisten. Da diese beiden Elemente aber weiterhin außerhalb des `GtkColumnView`-Widgets generiert werden sollten (sowohl der Inhalt der Fenster als auch die Funktionalität der Buttons erfordert Daten, auf die das `GtkColumnView`-Widget keinen Zugriff hat), muss die Erzeugung der Bildlaufleisten ebenfalls außerhalb stattfinden.

3.1.3 Aufbau

Bei dem entwickelten `GtkColumnView`-Widget handelt es sich um ein so genanntes „Composite-Widget“, also ein Widget, das aus anderen Widgets zusammengesetzt ist. Der Aufbau ist in Abbildung 3.1 zu sehen.

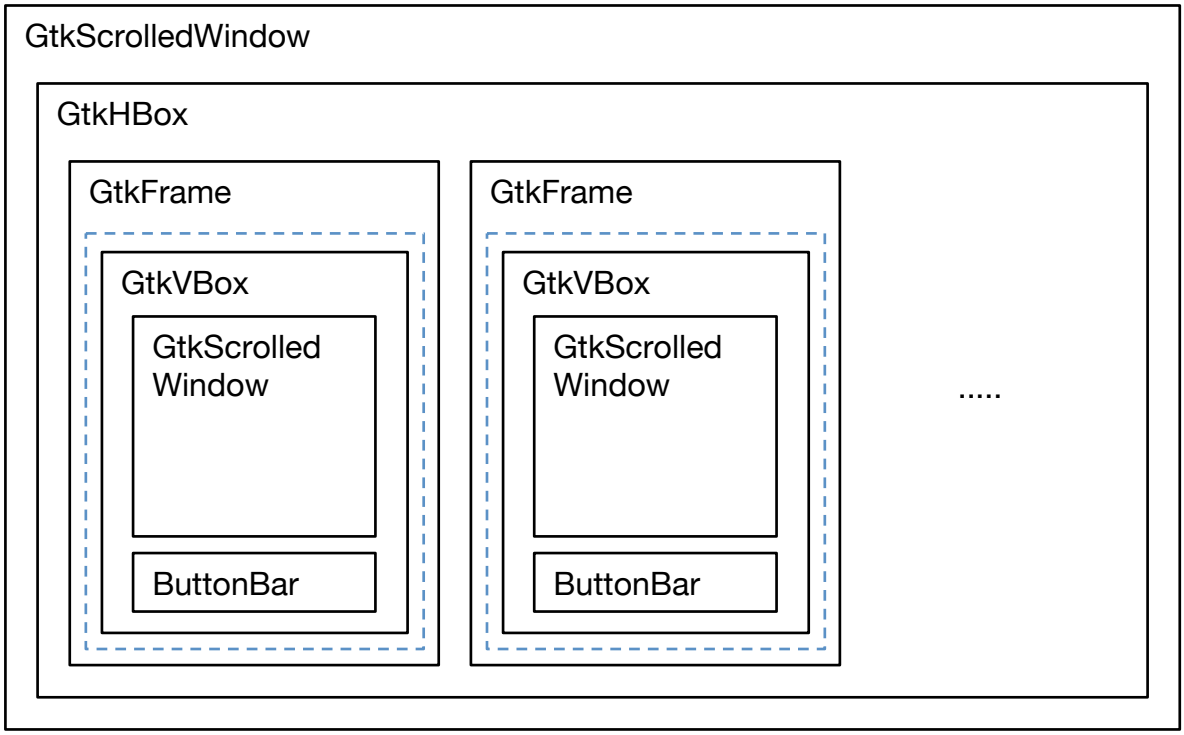

Abbildung 3.1: Aufbau des ColumnView-Widgets

Für jedes eingebettete Konfigurationsfenster wird ein GtkFrame (ein dünner Rahmen mit Titelzeile) generiert, um den Titel des Fensters anzeigen zu können und die Fenster visuell voneinander zu trennen. Die Konfigurationsfenster werden nebeneinander in einem

GtkHBox-Widget angeordnet, das beim Hinzufügen und Entfernen von Elementen dynamisch mitwächst und -schrumpft. Um das Scrollen des Inhalts zu ermöglichen, wird diese Struktur von einem GtkScrolledWindow-Widget umschlossen. Dieses Widget ist hier so konfiguriert, dass die horizontale Bildlaufleiste dauerhaft sichtbar ist. Wäre das nicht der Fall, so würde dieser erst beim Überschreiten der maximal sichtbaren Fensterbreite erscheinen, den Fensterinhalt stauchen und dabei Steuerelemente vertikal verlagern, was den Bedienkomfort vermindern würde.

Die Bereiche innerhalb der gestrichelten, blauen Rahmen in Abbildung 3.1 stellen die Änderungen an der GUI außerhalb des GtkColumnView-Widgets dar - der eigentliche Inhalt der Konfigurationsfenster wird nochmals in ein GtkScrolledWindow-Widget eingebettet, um ein unabhängiges, vertikales Scrollen zu ermöglichen.

3.1.4 Implementierungsdetails

Eine wesentliche Aufgabe des GtkColumnView-Widgets besteht darin, beim Erzeugen und Entfernen von Fenstern in der Spaltenansicht das jeweils aktiv gewordene Fenster sichtbar zu halten. Dafür genügt es, den horizontalen Bildlaufbalken bei jeder Änderung der Fensterstruktur nach rechts zu setzen, da dort immer das aktive Fenster zu finden ist.

Die einfachste Methode dafür wäre eine direkte Verarbeitung der Ereignisse beim Hinzufügen und Entfernen von Komponenten aus dem GtkHBox-Widget. Aus Gründen der Laufzeit-Datenhaltung werden geschlossene Fenster in der AnT 4.669-GUI aber nicht entfernt, sondern lediglich verborgen, was diese Möglichkeit effektiv ausschließt. Stattdessen werden hier die Signale genutzt, die von GTK+ verwendet werden, um Größenänderungen an Steuerelementen zu propagieren.

Möchte ein Steuerelement seine Größe ändern, so schickt es eine `"size-request"`-Anfrage an das umschließende Container-Widget, das daraufhin seine eigenen Größenanforderungen neu berechnen und seinerseits eine `"size-request"`-Anfrage an das Widget der nächsthöheren Ebene schicken kann, bis eine Anfrage das Widget der höchsten Ebene[2] erreicht. Dieses kann nun zusätzlichen Platz zur Verfügung stellen und mittels einer `"size-allocate"`-Anfrage auf die die direkt untergeordneten Widgets verteilen, die ihrerseits den ihnen zugeteilten Platz auf ihre untergeordneten Widgets verteilen können.

GTK+ bietet die Möglichkeit, solche Anfragen abzufangen und selbst zu verarbeiten. Dieser Mechanismus wird hier dazu genutzt, um zu erkennen, wann sich die effektive Größe des GtkHBox-Widgets verändert hat, indem alle `"size-allocate"`-Anfragen an dieses überwacht werden. Wird ein solches Signal empfangen, so wird die Callback-Funktion[3] `gtk_column_view_scroll_right(...)` aufgerufen, die die Position des Bildlaufbalkens an den rechten Rand setzt (siehe Listing 3.2).

[2] Im vorliegenden Fall ist dies bereits das GtkScrolledWindow, da es nahezu unbegrenzt Platz zur Verfügung stellen kann.

[3] Eine Callback-Funktion ist eine üblicherweise vom Benutzer eines Frameworks definierte Funktion, die das Framework dazu verwendet, um den Benutzer über aufgetretene Ereignisse zu benachrichtigen. Beispielsweise lässt sich für Buttons eine Callback-Funktion zum Ereignis `"pressed"` einrichten.

Listing 3.1 Automatisches Scrollen

```
static void
gtk_column_view_scroll_right
(GtkContainer* container, GtkAllocation* allocation, gpointer data)
{
  // get the horizontal scrollbar adjustment from the signal data
  GtkAdjustment* hAdj = gtk_scrolled_window_get_hadjustment (GTK_SCROLLED_WINDOW (data));

  // fail-safe
  if (hAdj == NULL) return;

  // scroll to the far right
  hAdj->value = hAdj->upper - hAdj->page_size;

  // notify the toolkit about the changes which then redraws the contents if necessary
  gtk_adjustment_value_changed (hAdj);
}

static void
gtk_column_view_init
(GtkColumnView *cv)
{
.
.
  gtk_signal_connect_after (GTK_OBJECT (cv->hBox),
                            "size-allocate",
                            GTK_SIGNAL_FUNC (gtk_column_view_scroll_right),
                            cv);
}
```

3.2 Integration des GtkColumnView-Widgets in AnT 4.669

3.2.1 FrameManager

Zur leichteren Handhabung und Integration des `GtkColumnView`-Widgets wurde als Interface die Klasse `FrameManager` entwickelt. Diese Klasse ist rein statisch definiert, es werden also keine Instanzen davon abgeleitet. Sie besteht aus einer Klassenvariablen `columnView` und drei Methoden, um auf diese zuzugreifen.

Die Methode
`FrameManager::reset()`
löscht das `GtkColumnView`-Widget, das durch die Variable `columnView` referenziert wird. Existiert noch kein solches Widget, so tut die Methode nichts.

Die Methode
`FrameManager::attach(GtkContainer *container)`
erzeugt ein `GtkColumnView`-Widget, fügt es in den zu übergebenden GtkContainer[4] ein und

[4]GtkContainer ist die Oberklasse aller Widgets, die andere Widgets enthalten können, wie zum Beispiel `GtkHBox` oder `GtkScrolledWindow`.

speichert eine interne Referenz zur späteren Verwendung in der Variablen `columnView`. Sollte schon ein `GtkColumnView`-Widget unter Verwaltung sein, wird es zunächst mittels der Methode `FrameManager::reset()` gelöscht, bevor das neue erstellt wird. Näheres zur Verwendung dieser Methode in Kapitel 3.2.2.

Die Methode
`FrameManager::getNewFrame(const char* title)`
hat zwei Aufgaben: Wird sie aufgerufen, bevor der `FrameManager` durch `FrameManager::attach(GtkContainer *container)` initialisiert wurde, erzeugt sie ein leeres Top-Level-Fenster[5] mit dem übergebenen Titel und liefert eine Referenz darauf zurück. Dieses Verhalten wird in der Initialisierungsphase dazu genutzt, um das Hauptfenster von AnT 4.669 zu erzeugen. Existiert zum Zeitpunkt des Aufrufs bereits ein verwaltetes `GtkColumnView`-Widget, so wird stattdessen eine neue Spalte darin generiert und die Referenz darauf zurückgegeben. Die neue Spalte wird rechts neben die schon vorhandenen angefügt. Näheres zur Verwendung dieser Methode in Kapitel 3.2.3.

Listing 3.2 Klassendeklaration FrameManager

```
class FrameManager
{
private:
  static GtkWidget* columnView;

public:
  static void attach (GtkContainer* container);
  static void reset ();

  static GtkWidget* getNewFrame (const char* title);
};
```

3.2.2 Initialisierungsphase

Beim Start von AnT 4.669 wird das Hauptfenster der Anwendung im Konstruktor der Klasse `MainWindow` erzeugt. Im rechten Bereich wird nun außer dem Logo (siehe Abbildung 3.2) auch ein Platzhalter-Widget generiert, in den mit `FrameManager::attach(...)` das `GtkColumnView`-Widget eingebettet wird. Zudem wurden Höhe und Breite des Hauptfensters vergrößert, um mehr Platz für die Konfigurationsfenster zu bieten und vertikales Scrollen zu vermeiden. Zuvor wurde die Größe des Hauptfensters indirekt durch die Größe des eingebetteten Logos bestimmt, nun wird lediglich die Initialgröße durch die beiden Konstanten `MAIN_WINDOW_INITIAL_WIDTH` und `MAIN_WINDOW_INITIAL_HEIGHT`[6] festgelegt, ein Vergrößern des des Fensters während der Laufzeit wurde ermöglicht.

[5] Top-Level-Fenster sind Fenster, die vom Betriebssystem/Fenstermanager mit Rahmen und Titelleiste ausgestattet werden.

[6] Definiert in Der Datei `src/ant-gui/Settings.hpp` ab Zeile 65

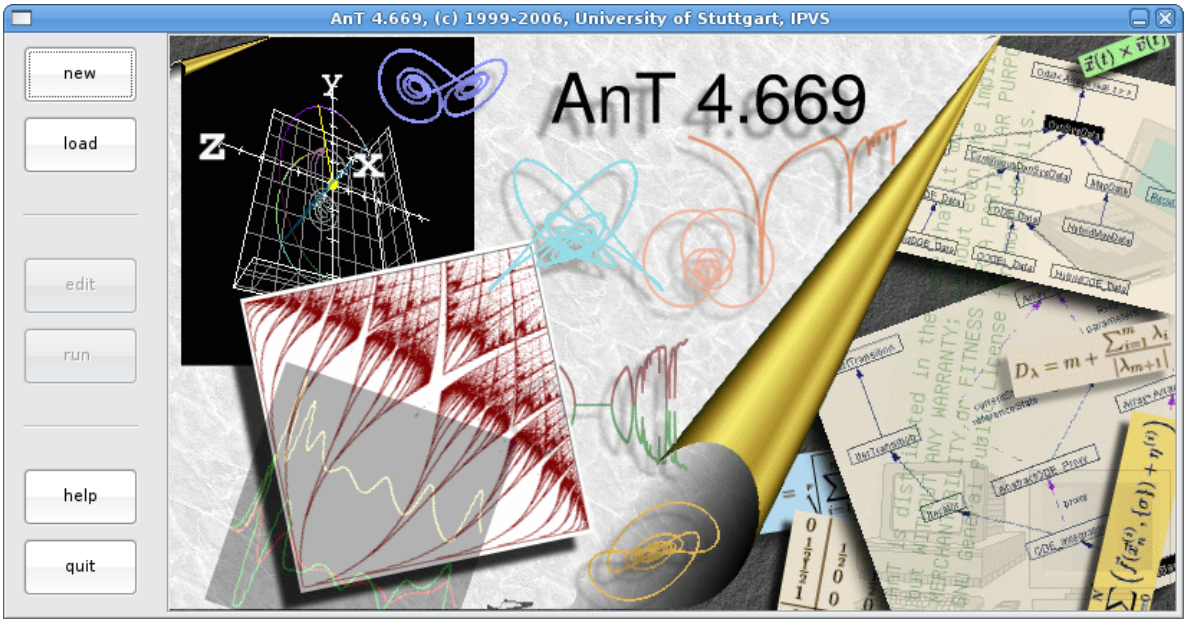

Abbildung 3.2: Ursprüngliches Hauptfenster von AnT 4.669

3.2.3 Generierung von Fenstern

Neue Fenster werden in der AnT 4.669-GUI durch Aufruf der globalen Funktion `getDefaultToplevelWindow(const char* title)` generiert, die als Parameter den Titel des Fensters erwartet und eine Referenz auf das Fenster-Widget zurück gibt. Diese zentralisierte Methode erleichterte die Umstellung auf die Spaltenansicht an dieser Stelle: Wie in Listing 5.2 zu sehen ist, konnte diese Funktion unter Verwendung von `FrameManager::getNewFrame(...)` auf eine Zeile reduziert werden, da der entsprechende Code nun in dieser Methode steht.

3.2.4 Anpassungen

Wie schon in Abschnitt 3.1.2 erwähnt, brachte die Umstellung auf die Spaltenansicht eine einheitliche Höhe für die Konfigurationsfenster mit sich. Für eine übersichtliche Darstellung der Steuerelemente mussten einige Anpassungen vorgenommen werden. Zunächst wurde die Buttonleiste der Konfigurationsfenster am unteren Ende des Fensters verankert und der „quit“-Button entfernt, da für diese Funktion nun der entsprechende Button des Hauptfensters genutzt werden kann. Der Bereich oberhalb der Buttonleiste wurde wie in Abschnitt 3.1.3 beschrieben in ein `GtkScrolledWindow`-Widget eingebettet, um ein unabhängiges Scrol-

Listing 3.3 getDefaultTopLevelWindow() - neuer Code zwischen #ifdef und #else, alter Code zwischen #else und #endif

```
GtkWidget* getDefaultToplevelWindow (const char* title)
{
#ifdef IN_PLACE_CONFIGURATOR
  return FrameManager::getNewFrame (title);
#else
  GtkWidget* result = gtk_window_new (GTK_WINDOW_TOPLEVEL);

  gtk_widget_set_name (result, title);
  gtk_window_set_title (GTK_WINDOW (result), title);

  gtk_window_set_policy ( GTK_WINDOW (result),
                          FALSE, /* allow_shrink */
                          FALSE, /* allow_grow */
                          TRUE /* auto_shrink */ );

  gtk_window_set_position ( GTK_WINDOW (result),
                            GTK_WIN_POS_MOUSE );

  return result;
#endif
}
```

len von sehr langen Konfigurationsfenstern zu ermöglichen[7], ohne eine globale vertikale Bildlaufleiste erstellen zu müssen (siehe Spalte „window[]“ in Abbildung 3.3).

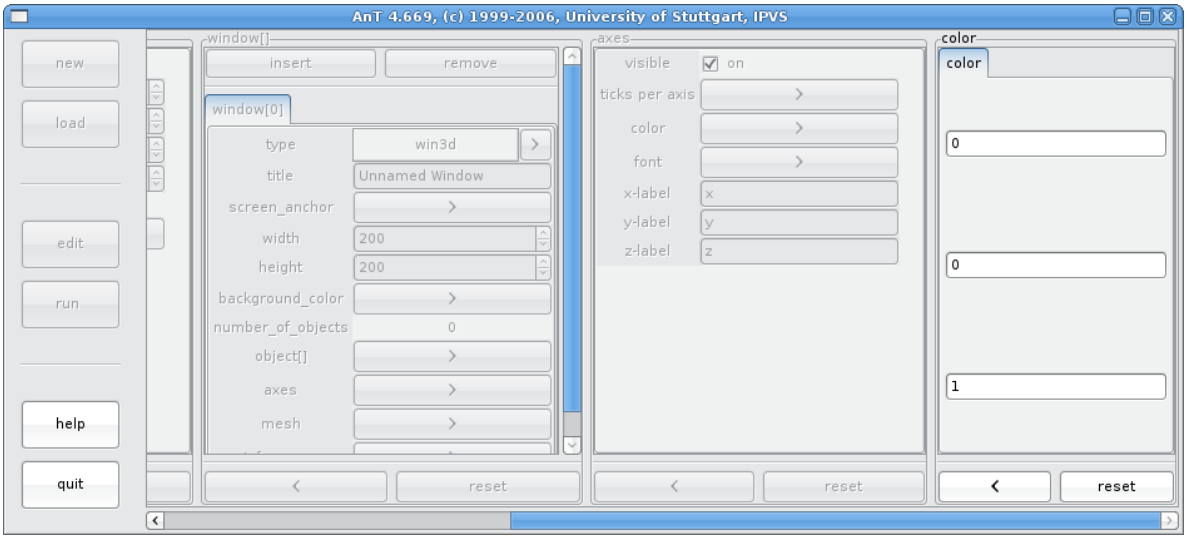

Abbildung 3.3: Neues Hauptfenster von AnT 4.669, zu Demonstrationszwecken Höhe verringert und Änderungen im Abstand der Steuerelemente rückgängig gemacht

[7]In der aktuellen Version von AnT 4.669 wird dieses Feature normalerweise nicht benötigt, zu Demonstrationszwecken wurde daher die Höhe des Fensters in Abbildung 3.3 verringert.

Die an vielen Stellen des ursprünglichen Quellcodes genutzte Funktion `gtk_container_add(...)` wurde durch `gtk_box_pack_start(...)` ersetzt, da erstere die Steuerelemente über die gesamte verfügbare Höhe des Fensters verteilt (siehe Spalte „color“ in Abbildung 3.3), während letztere alle Elemente am oberen Ende gruppieren kann. Ein Beispiel hierzu:

Listing 3.4 Ausschnitt aus der Quelldatei Configurators.cpp ab Zeile 3325 - neuer Code zwischen #ifdef und #else, alter Code zwischen #else und #endif

```
#ifdef IN_PLACE_CONFIGURATOR
    gtk_box_pack_start (GTK_BOX (result), buttonBox, FALSE, FALSE, 0);

    gtk_box_pack_start (GTK_BOX (result), gtk_hseparator_new (), FALSE, FALSE, 0);
#else
   gtk_container_add ( GTK_CONTAINER (result),
                       buttonBox );

   gtk_container_add ( GTK_CONTAINER (result),
                       gtk_hseparator_new () );
#endif
```

Die beiden `"FALSE"`-Parameter in den `gtk_box_pack_start(...)`-Aufrufen weisen die Containerklasse `GtkVBox` an, überschüssigen Platz nicht unter den Steuerelementen aufzuteilen, sondern ihnen nur die minimal nötige Höhe zuzuweisen.

In der Mehrfenstervariante wurde stets das gesamte Hauptmenü deaktiviert, sobald ein weiteres Fenster geöffnet wurde. Dieses Verhalten wurde nun angepasst: Wird einer der Punkte „new“, „edit“ oder „run“ ausgewählt, so wird nicht mehr alle Menüpunkte deaktiviert: Die beiden Buttons „help“ und „quit“ werden stets aktiv gehalten, damit sie auch bei geöffneten Konfigurationsfenstern verwendet werden können.

Durch die Vergrößerung des Hauptfensters und die Nutzung des Raums, der ursprünglich nur dem Logo vorbehalten war, musste dieses neu zentriert werden. Zudem wird das Logo nun verborgen, sobald das erste Konfigurationsfenster geöffnet wird. Weiterhin ist nun die Möglichkeit vorgesehen, aus mehreren verfügbaren Logos nach dem Zufallsprinzip eines auszuwählen und anzuzeigen. Der zusätzlich gewonnene Platz unterhalb des Logos wird dabei dazu genutzt, optionale Bildunterschriften einzublenden, die zusammen mit dem Logo gespeichert werden.

3.3 Neue und veränderte Quellcode-Dateien

Im folgenden eine kurze Übersicht über veränderte und neu erstellte Dateien im AnT 4.669-Quellcode. Verwaltungsdateien (wie zum Beispiel Makefiles) und Logos sind hier nicht aufgeführt.

3.3.1 Settings.hpp

In dieser Datei wird nun die initiale Größe des Fensters festgelegt, außerdem werden hier in Zeile 64 alle Änderungen, die im Laufe dieser Arbeit entstanden sind aktiviert. Für die ursprüngliche Version der GUI kann diese Zeile auskommentiert werden.

Listing 3.5 Ausschnitt aus der Quelldatei Settings.hpp ab Zeile 64

```
...
#define IN_PLACE_CONFIGURATOR
#define MAIN_WINDOW_INITIAL_WIDTH 800
#define MAIN_WINDOW_INITIAL_HEIGHT 600
...
```

3.3.2 FrameManager.hpp und FrameManager.cpp

Diese beiden Dateien sind neu hinzugekommen. In ihnen werden sowohl das `GtkColumnView`-Widget als auch die Klasse `FrameManager` definiert und implementiert.

3.3.3 GeneralWidgets.cpp

In dieser Datei wurde die ursprüngliche Funktion zum Öffnen von neuen Fenstern so angepasst, dass sie den `FrameManager` verwendet.

3.3.4 MainWindow.hpp und MainWindow.cpp

Hier sind alle Änderungen zu finden, die die Initialisierungsphase, das Hauptfenster (Layoutänderungen, Logos und die Einbettung des `GtkColumnView`-Widgets) und das Hauptmenü betreffen.

3.3.5 Configurators.cpp

In dieser Datei wurden die Anpassungen am Layout der Konfigurationsfenster vorgenommen.

3.3.6 SemanticCheck.cpp

Hier wurde die Methode, die beim Speichern einer Konfigurationsdatei aufgerufen wird, so angepasst, dass sie nach dem Speichervorgang die korrekten Buttons im Hauptmenü aktiv schaltet.

Part II

A Brief Introduction to AnT 4.669

Chapter 4

Introduction

Using AnT 4.669 can be confusing at first because of the great number of options it offers. Therefore I decided to write this tutorial to help you on your first steps with AnT 4.669.

4.1 Things you will need

In order to follow the steps described in the tutorial, you will need have the following:

- Linux or another UNIX derivative, with a running X server and a shell terminal
- AnT 4.669, obviously `http://www.ant4669.de`
- gnuplot `http://www.gnuplot.info`

It is worth noting that AnT 4.669 is also available on Windows and Mac OS X, but this tutorial contains steps specific to the UNIX platform.

Chapter 5

First steps

We are going to use the logistic map as our first example. It is advisable to create a new directory for every system you want to analyze using AnT 4.669 to keep things tidy. Also the files that AnT 4.669 creates have default names which you don't have to change that way.

5.1 The system function

The logistic map's system function is:

$x_{n+1} = rx_n(1 - x_n)$

There are two options to let AnT 4.669 know about this function. You can use the builtin interpreter by entering the function in the GUI as we will do at first or you can implement the function in C++, compile it as a library and let AnT 4.669 use it. While the former is easier to understand and faster to set up, the latter is more flexible, usually faster in execution and is thus preferred when speed is an issue.

In the following section we will first use the AnT 4.669 interpreter. The precompiled library will be covered in section 5.4.

5.2 Setting up the configuration file

AnT 4.669 is driven by configuration files containing instructions which the user creates. For that purpose we will use the supplied GUI. Start the GUI by typing `"AnT-gui"` in your terminal. You should see the following screen[1]:

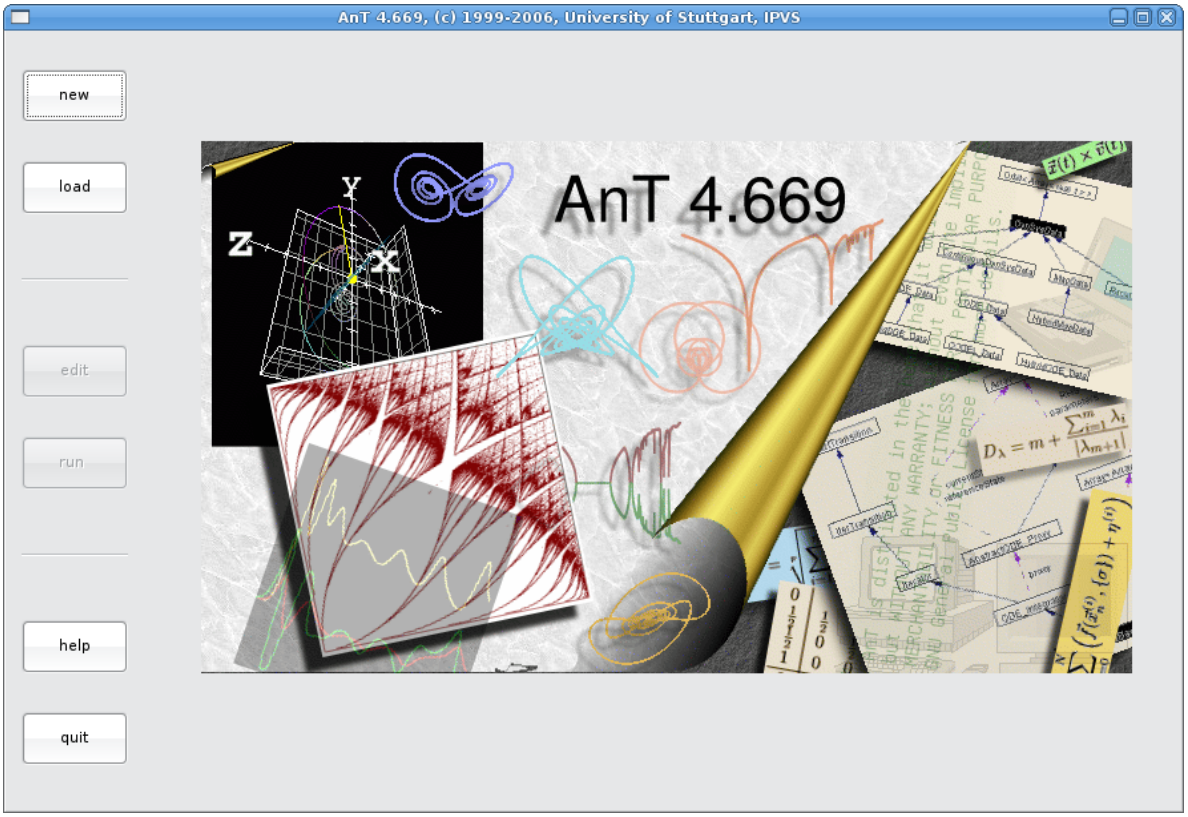

Figure 5.1: AnT 4.669 main window

You could now click “load” to open an existing configuration file, but as we want to create one from scratch, just click “new”.

[1]The logo may vary, as it is randomly selected from a set of logos.

You are now at the root configuration node. Choose “dynamical system” to configure the system to be analyzed.

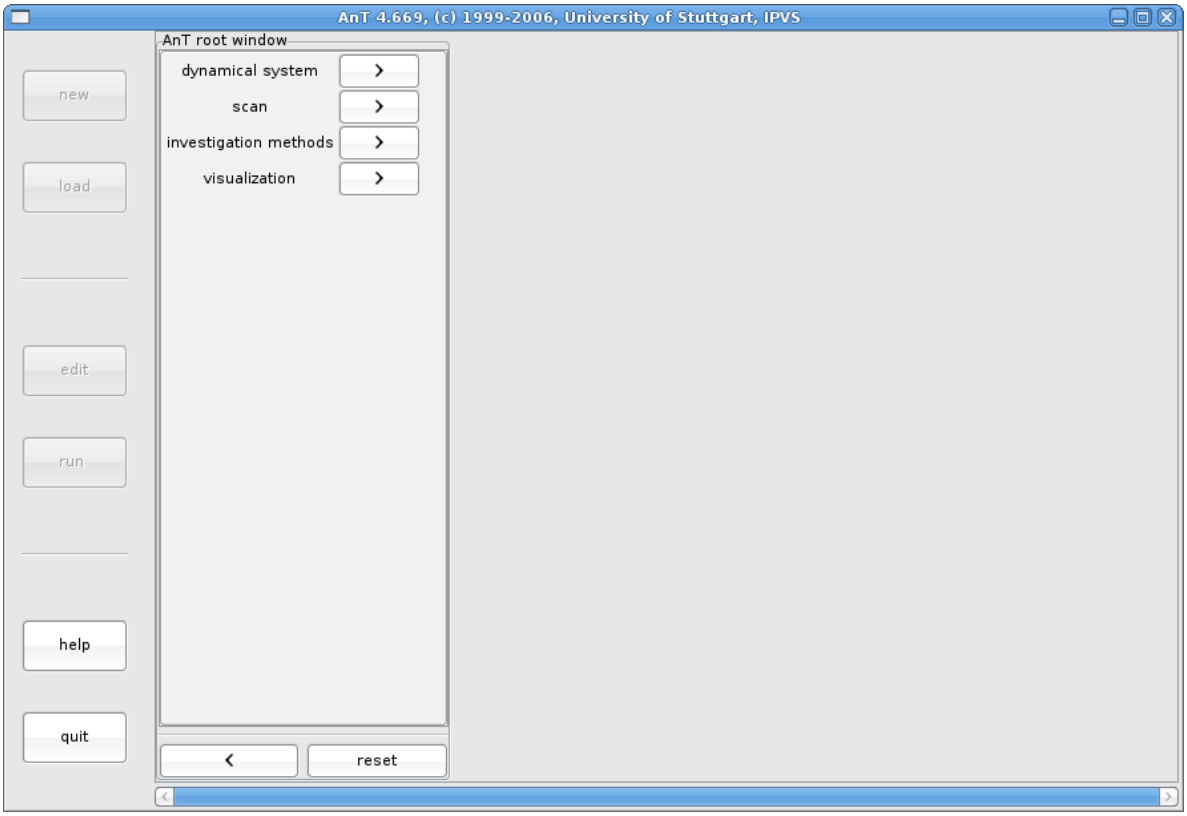

Figure 5.2: Root configuration node

Since the logistic map is a map type dynamical system, we don't need to change the default value of the "dynamical system type" option. When dealing with other system types, for example ordinary or delayed differential equations, this is where you tell AnT 4.669 about it.

Enter the name of the system, in our case "logistic". This step is optional, but should be taken to avoid possible confusion when editing the configuration file later. As the logistic map is a scalar system, its state space dimension is 1. You can increase the number of iterations to be able to detect attractors that converge very slowly, but for a first glimpse the default value should suffice.

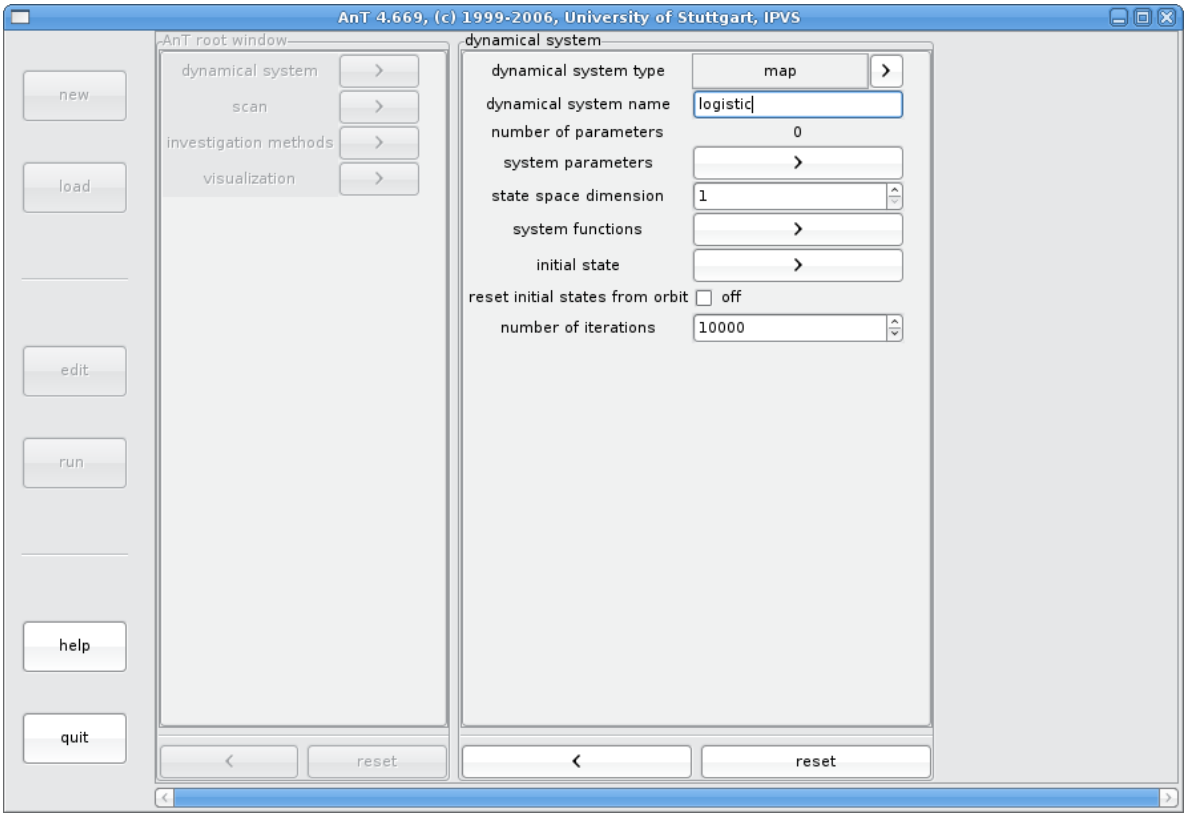

Figure 5.3: Dynamical system settings

Now choose "system parameters" to specify the parameters the logistic map contains.

We now have the choice to handle the parameters either via an array or separately. For the logistic map with its single parameter, the latter is the obvious choice. Click on “specific parameters” to continue.

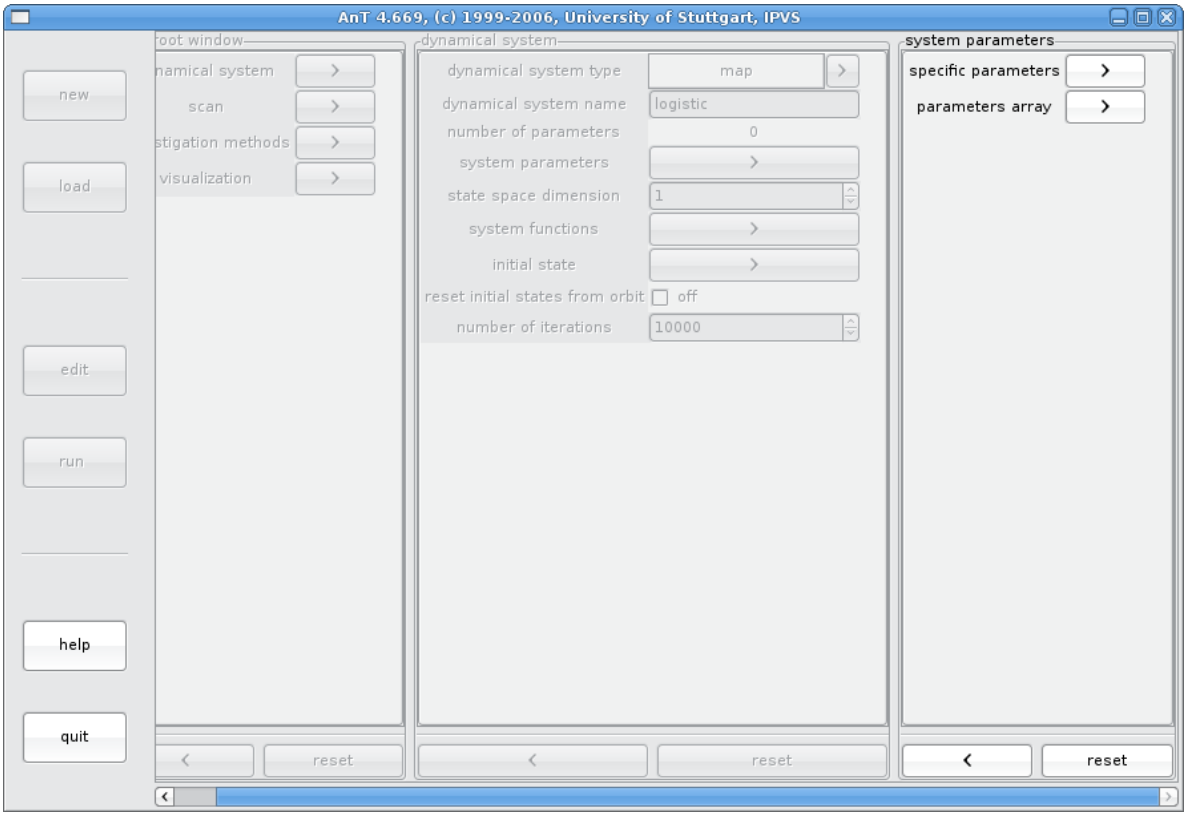

Figure 5.4: System parameters

In this window, click “insert” once to create a new parameter template. Enter “r” as the name. This name will be used when entering the system function for the interpreter and it will be available as a scan object. Its value doesn’t matter for our purposes, as we will enter a range of values to be used for that parameter later. When investigating a system for a fixed set of parameters this is where you initialize them.

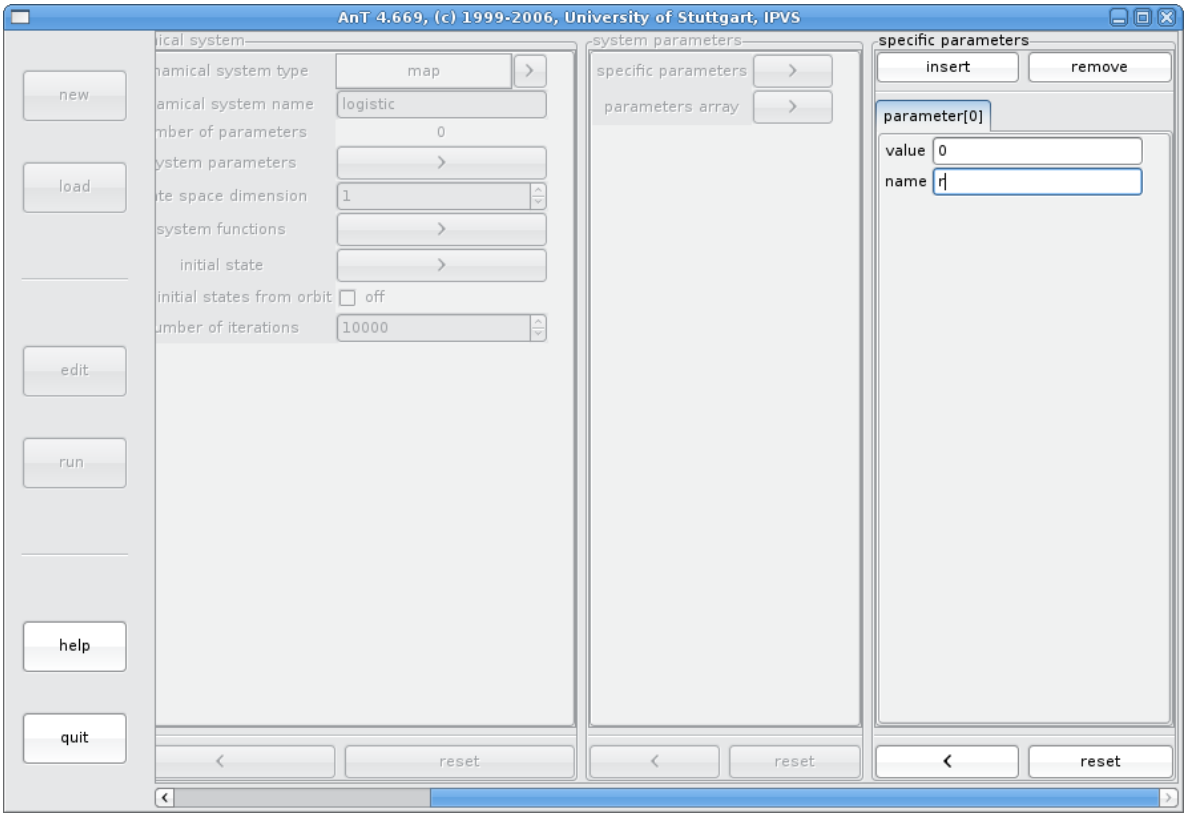

Figure 5.5: Specific parameters

Now close the two last windows by clicking the “<” button on the bottom to return to the main dynamical system window (Figure 5.3). From there, choose “system functions”.

Here we can supply the system function, if we want to use the builtin interpreter, which we currently do. Enter `"x"` as the name of the function and `"r*x*(1-x)"` as its equation of motion.

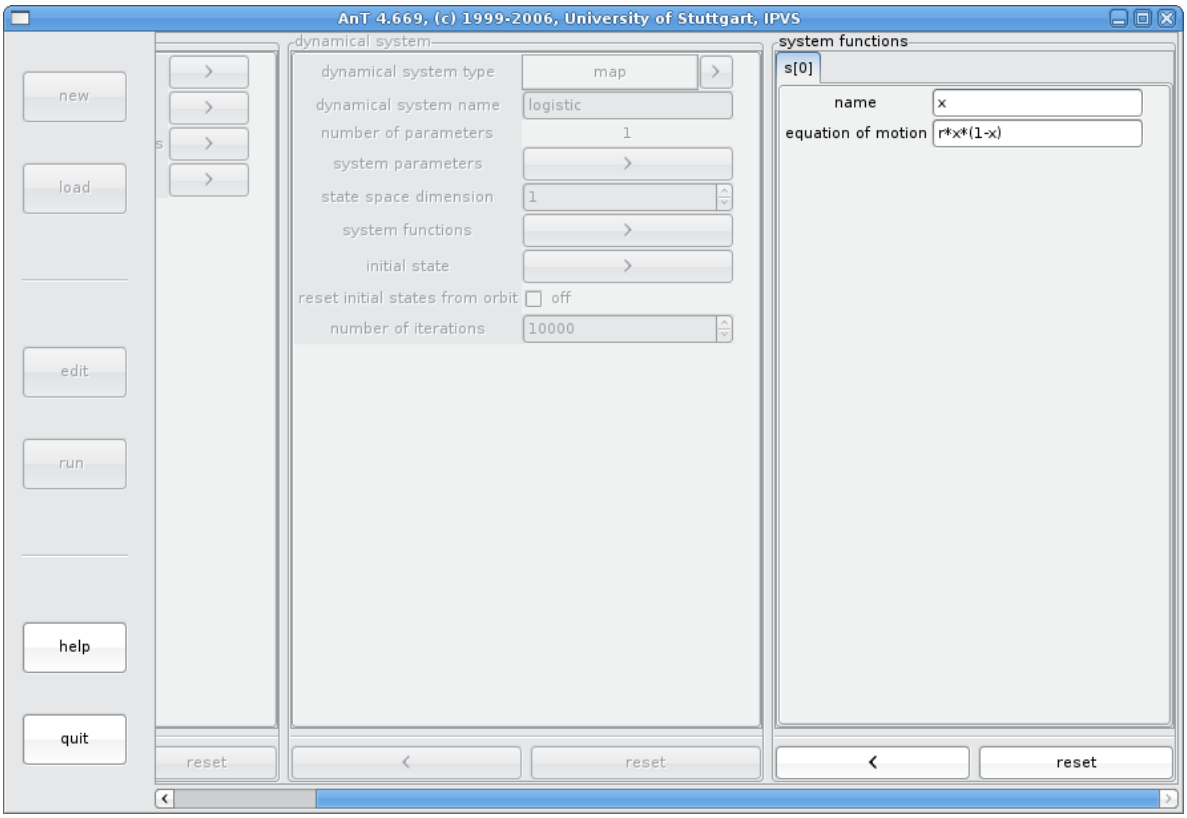

Figure 5.6: System function

After finishing, close the window and select “initial state”.

The value entered here is used by AnT 4.669 as the system state for the first iteration. The default value of "0" is not recommended, as it is a fixed point in the logistic map, regardless of the parameter r (stable for $r < 1$ and unstable for $r > 1$). In order to enable AnT 4.669 to find different attractors, set it to another value in the range (0;1].

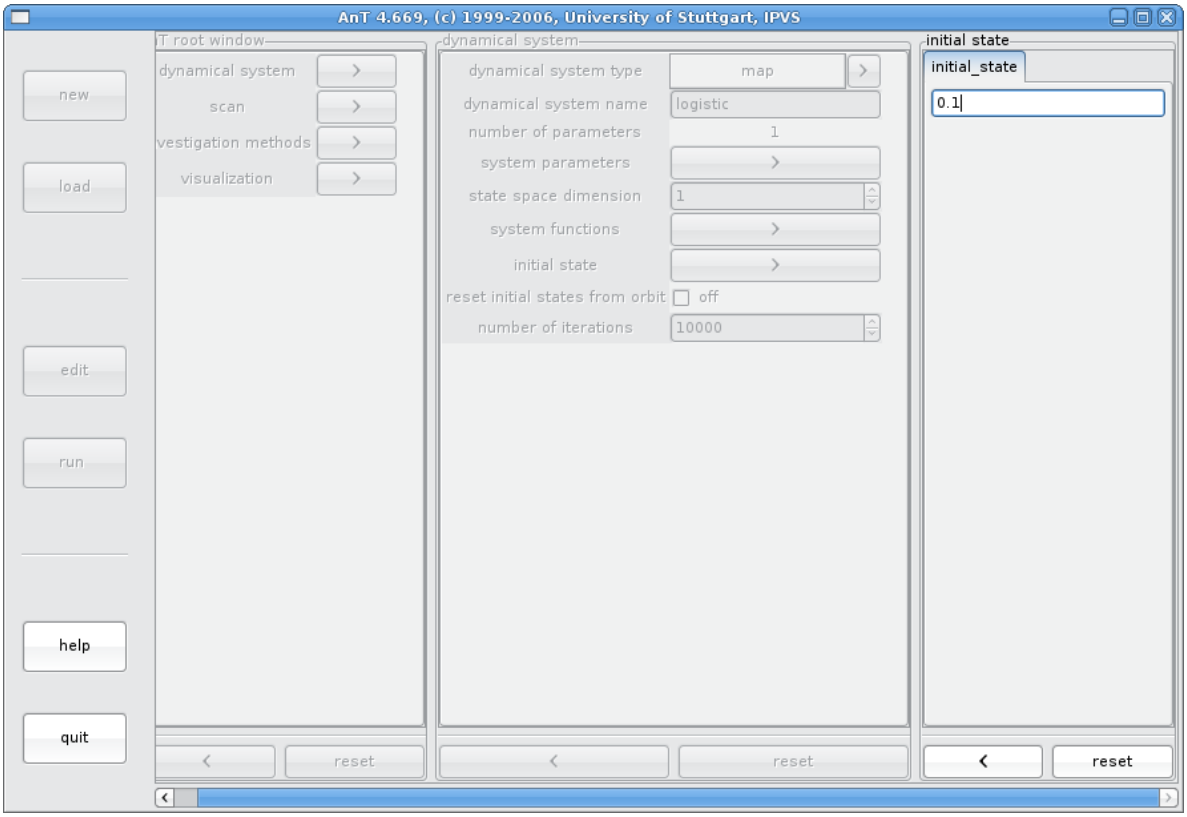

Figure 5.7: Initial system state

Close the last two windows to return to the root node. Then click "scan" to configure the scan parameters.

The “scan mode” entered here is basically the number of objects (for example system parameters or initial values) varied within the scan. In our example we are going to vary only one parameter (r), therefore enter “1” here.

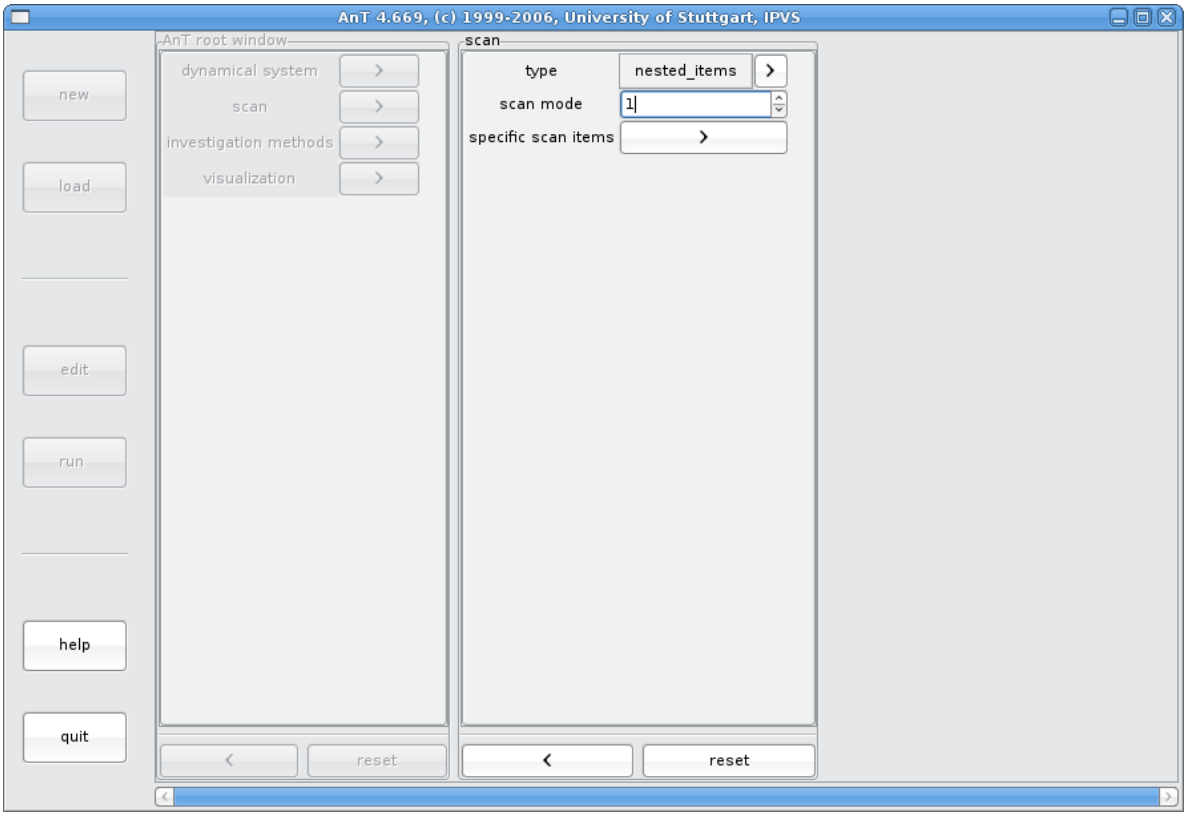

Figure 5.8: Scan settings

Click “specific scan items” to configure the parameter range.

Enter “0” and “4” as minimum and maximum values. The “points” option lets you choose the number of samples to be calculated in this interval. For this example, enter “1000”.

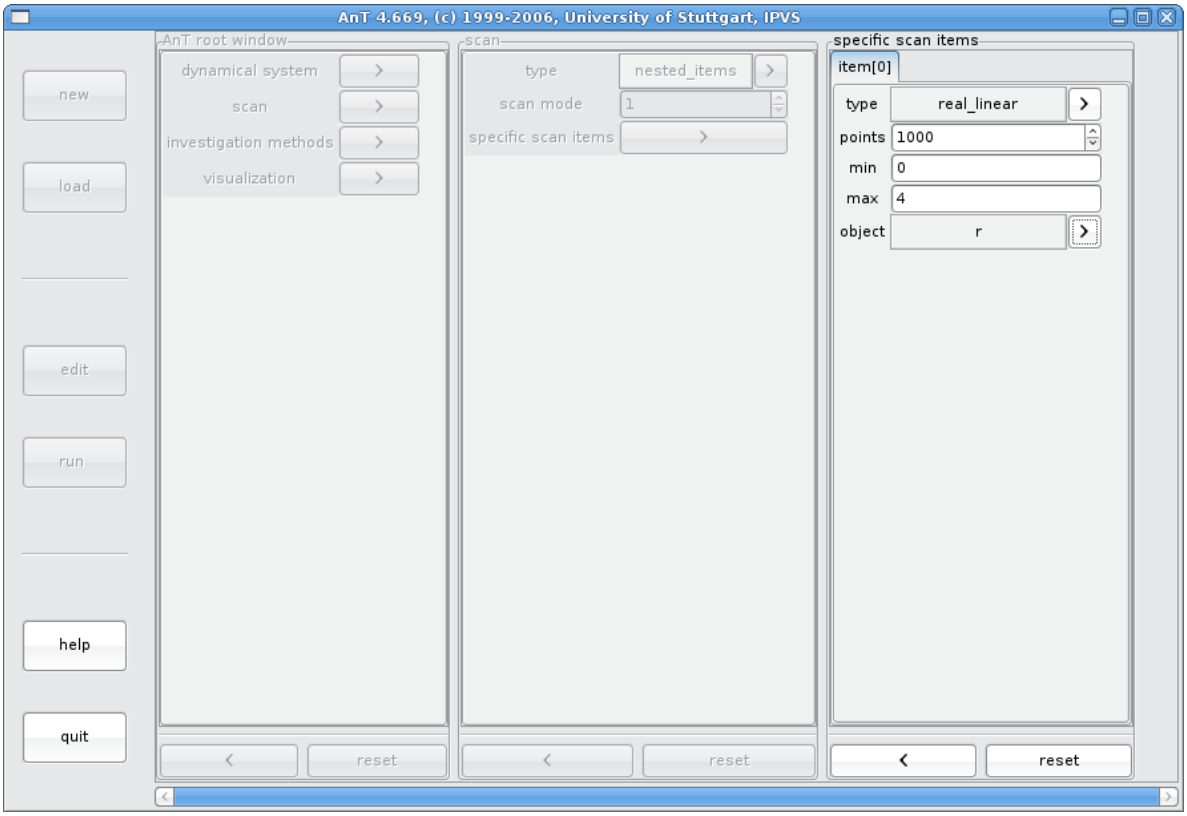

Figure 5.9: Scan range and resolution

Go back to the root node and choose “investigation methods” to tell AnT 4.669 what methods you are interested in.

Click on “period analysis” as we want to investigate periodic orbits for this tutorial.

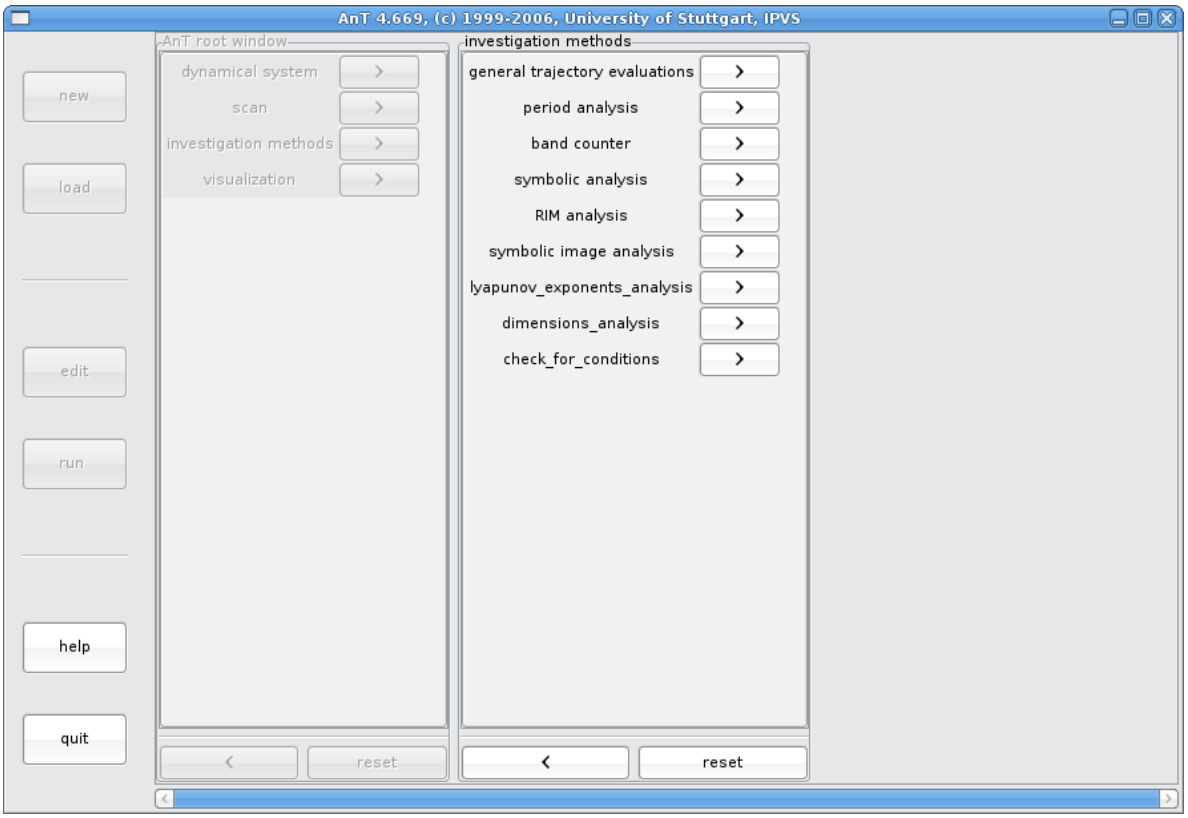

Figure 5.10: Investigation methods

Check the “is active” box to turn on this method. After that, check “period”, “cyclic asymptotic set” and “acyclic asymptotic set” and leave the related filenames as they are. These settings instruct AnT 4.669 to scan for periodic and aperiodic attractors. If AnT 4.669 finds a cycle in its last 128 (or whatever value you enter for “maximal period”) iterations (using the given compare precision), the periodic points and current parameter values are saved in “bif_cyclic.tna” and the period length in “period.tna”. In the case that no cycle is found before the maximum number of iterations is reached, the current parameter values are saved in “bif_acyclic.tna”.

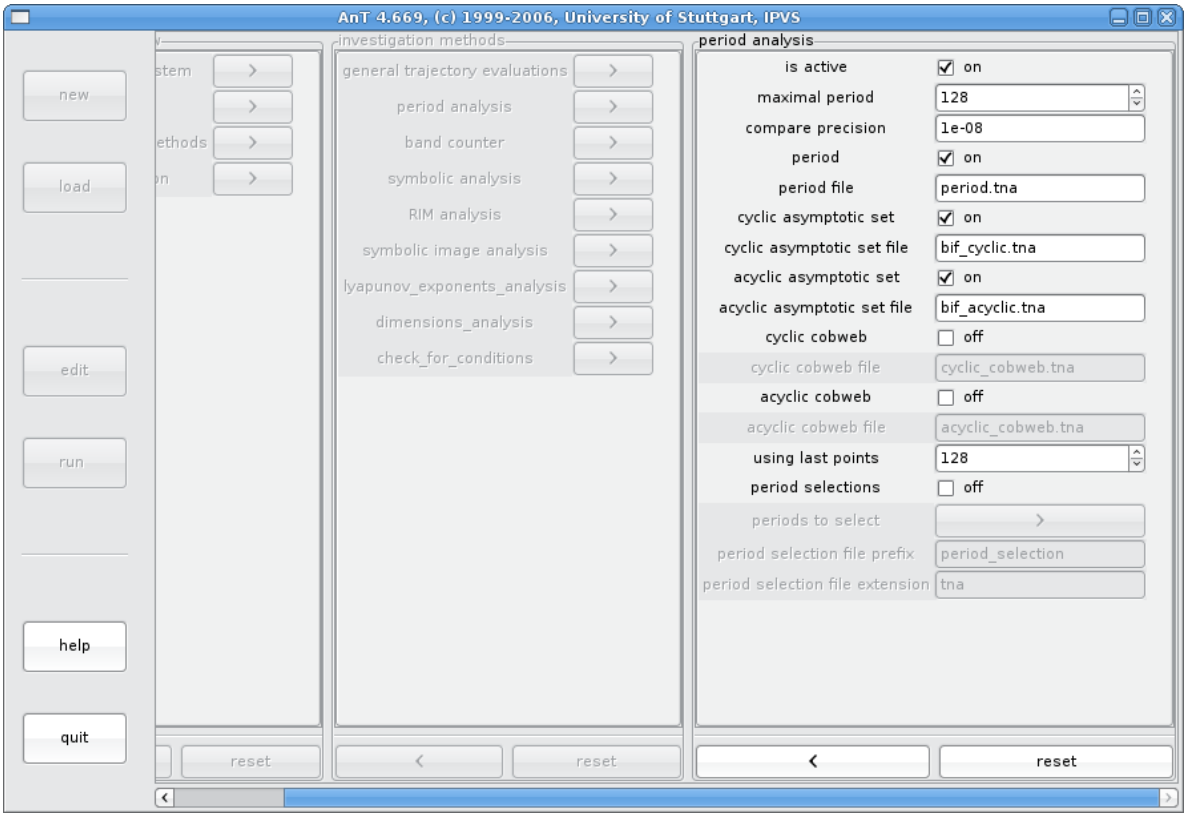

Figure 5.11: Period analysis settings

Close the period analysis window and select “lyapunov_exponent_analysis” from the methods menu.

Check the “is active” box to enable the numerical calculation of Lyapunov exponents. The default values are again reasonable for a first draft.

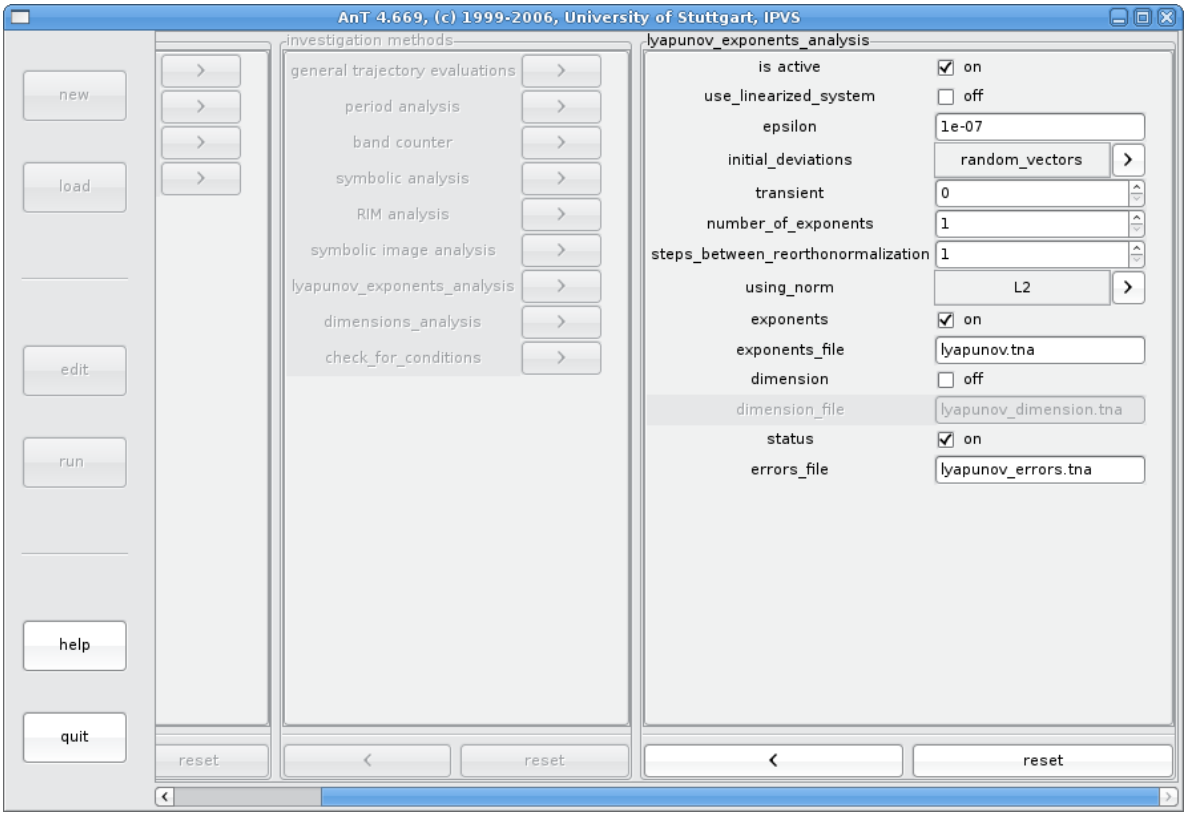

Figure 5.12: Period analysis settings

Now close all open configuration windows, including the root window. A save dialog should pop up, enter `"logistic.ant"` as the name and save.

5.3 Running AnT 4.669

There are two methods of running AnT 4.669. You can start AnT 4.669 via the the GUI or directly from the command line. We will discuss the former method first:

From the main menu, select “run”.

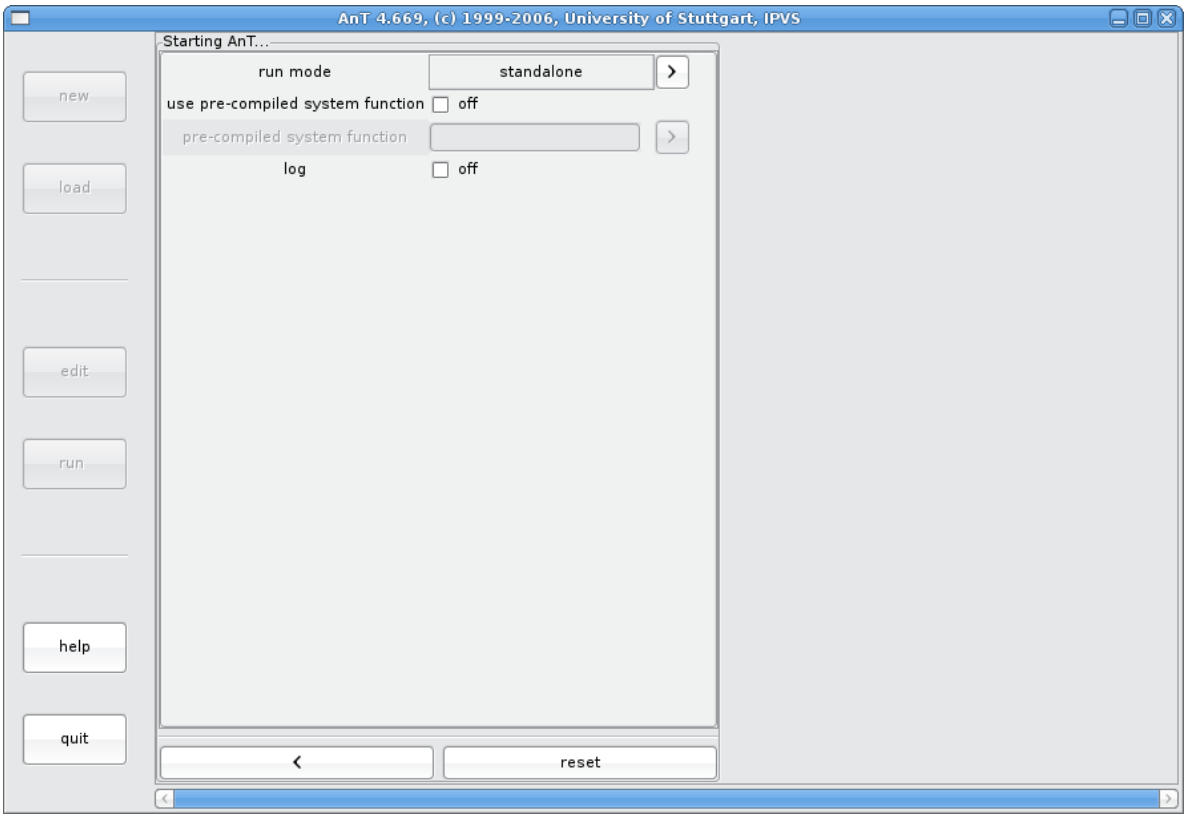

Figure 5.13: Run options

Leave all options in the appearing window at default and close the window again. The whole GUI should quit now and AnT 4.669 should start its scanning process with an output like this:

Listing 5.1 AnT 4.669 output

```
WARNING: string entry for 'pre-compiled system function' is missing!
Name of the dynamical system:
Name of the configuration file: /home/test/systems/map/logistic/logistic.ant
runmode: standalone
loading the system...
Using a parsed systemfunction...
Adding investigation method for key: 'period_analysis' ...
investigation method added successfully.
allocated size of the continuous orbit: 129
the following real-valued objects are scannable:
    initial_state[0][0]
    parameter[0]
    r
scannable objects inspected.
starting scanMachine...
total: 1000. calculated 10. progress 1 %.
total: 1000. calculated 20. progress 2 %.
total: 1000. calculated 30. progress 3 %.
total: 1000. calculated 40. progress 4 %.
total: 1000. calculated 50. progress 5 %.
total: 1000. calculated 60. progress 6 %.
total: 1000. calculated 70. progress 7 %.
total: 1000. calculated 80. progress 8 %.
total: 1000. calculated 90. progress 9 %.
total: 1000. calculated 100. progress 10 %.
```

Depending on your processor speed, it could take a while for this process to finish. The more values are scanned, the longer it takes.

To start AnT 4.669 from the command line, just type `"AnT -i ./logistic.ant"` in the system directory.

5.4 Using a precompiled system function

In order to speed up the calculation process, we should implement the logistic map as a C++ function. The name of the source file should be the same as the configuration file, except for the extension, which is `.cpp` here. This makes starting AnT 4.669 from the command line a bit easier[2]. Therefore we will call it `"logistic.cpp`. The first line in such a system definition should be `#include "AnT.hpp"`, which enables us to use the interface to AnT 4.669.

In this interface a map's system function is realized by a function of the form

```
bool sysname(const Array<real_t>& currentState,
             const Array<real_t>& parameters,
             Array<real_t>& RHS)
```

which simulates a step of the system. The first two array parameters hold the old state vector of the system and the system's parameters. After using those two values to calculate the new state vector, the function is supposed to write it into the array `RHS` and return true if all went well.

Note: In order to be able to write more readable code, you should define macros to assign names to those parameters.

Every time AnT 4.669 wants to calculate an iteration of the system, it calls `MapProxy::systemFunction`, a function pointer. We have to set this pointer to our newly written function to make it accessible to AnT 4.669, the opportunity to do so is during the initialisation phase, when the function `connectSystem()` gets called by AnT 4.669.

Listing 5.2 logistic.cpp

```
#include "AnT.hpp"

#define r (parameters[0])
#define Xn (currentState[0])
#define Xn1 (RHS[0])

bool logistic (const Array<real_t>& currentState,
               const Array<real_t>& parameters,
               Array<real_t>& RHS)
{
  Xn1 = r * X * (1 - X);

  return true;
}

extern "C"
{
  void connectSystem ()
  {
    MapProxy::systemFunction = logistic;
  }
}
```

[2] If the two file names don't match, we have to tell AnT about both of them.

After saving `logistic.cpp` we have to compile it into a library for AnT 4.669 to use. The easiest way to do this is to call the shell script `make-AnT-system.sh`, which gets installed along with AnT 4.669. This script needs no parameters, as it automatically scans all files in the current directory ending in `".cpp"` for the presence of the `connectSystem` function. If there were no errors in `logistic.cpp`, you should now also have a file called `"logistic.so"`. This is the library that AnT 4.669 needs to simulate the system.

5.5 Running AnT 4.669 using a precompiled system function

With a precompiled system function the process of starting AnT 4.669 is a bit different. In the GUI again choose “run” from the main menu.

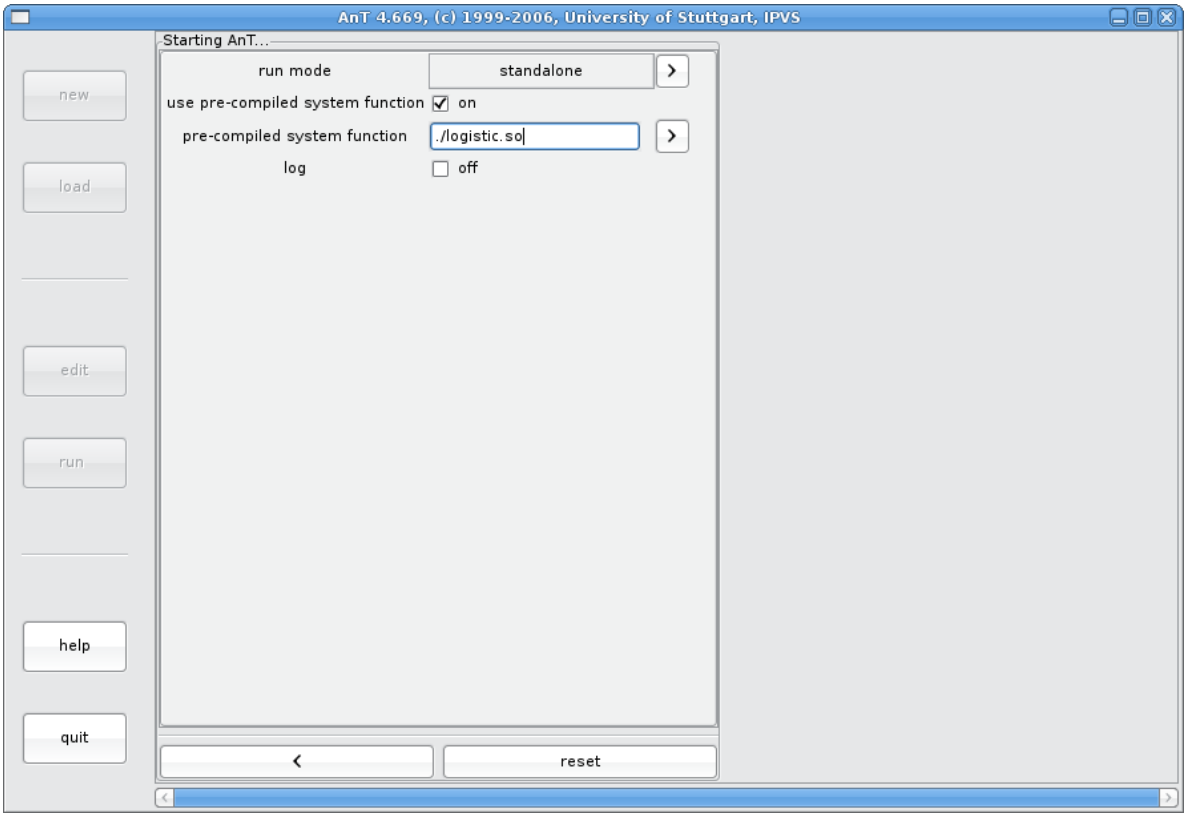

Figure 5.14: Run options for precompiled system function

This time, check the option “use pre-compiled system function” and enter `"./logistic.so"`[3] as the “pre-compiled system function”. Close the window and AnT 4.669 should again start scanning. Note the increase in speed achieved by precompiling.

Alternatively you can start AnT 4.669 from the command line again by typing `"AnT ./logistic"`[4] in the system directory.

[3] You can also use the file selector button to the right of the text field.

[4] Or `"AnT ./<precompiled> -i ./<config.ant>"` if the file names do not match – omit the library’s file extension in any case!

5.6 Displaying the results in gnuplot

After AnT 4.669 is finished, four new files should have been created: `"period.tna"`, `"bif_cyclic.tna"`, `"bif_acyclic.tna"` and `"lyapunov.tna"`. To visualize the values in those files, we use gnuplot. Start it by typing `"gnuplot"`. You should now see the gnuplot prompt. At that prompt, enter

```
gnuplot> plot 'bif_cyclic.tna' with dots, 'bif_acyclic.tna' with dots
```

to plot a bifurcation diagram. A new window should open, similar to this:

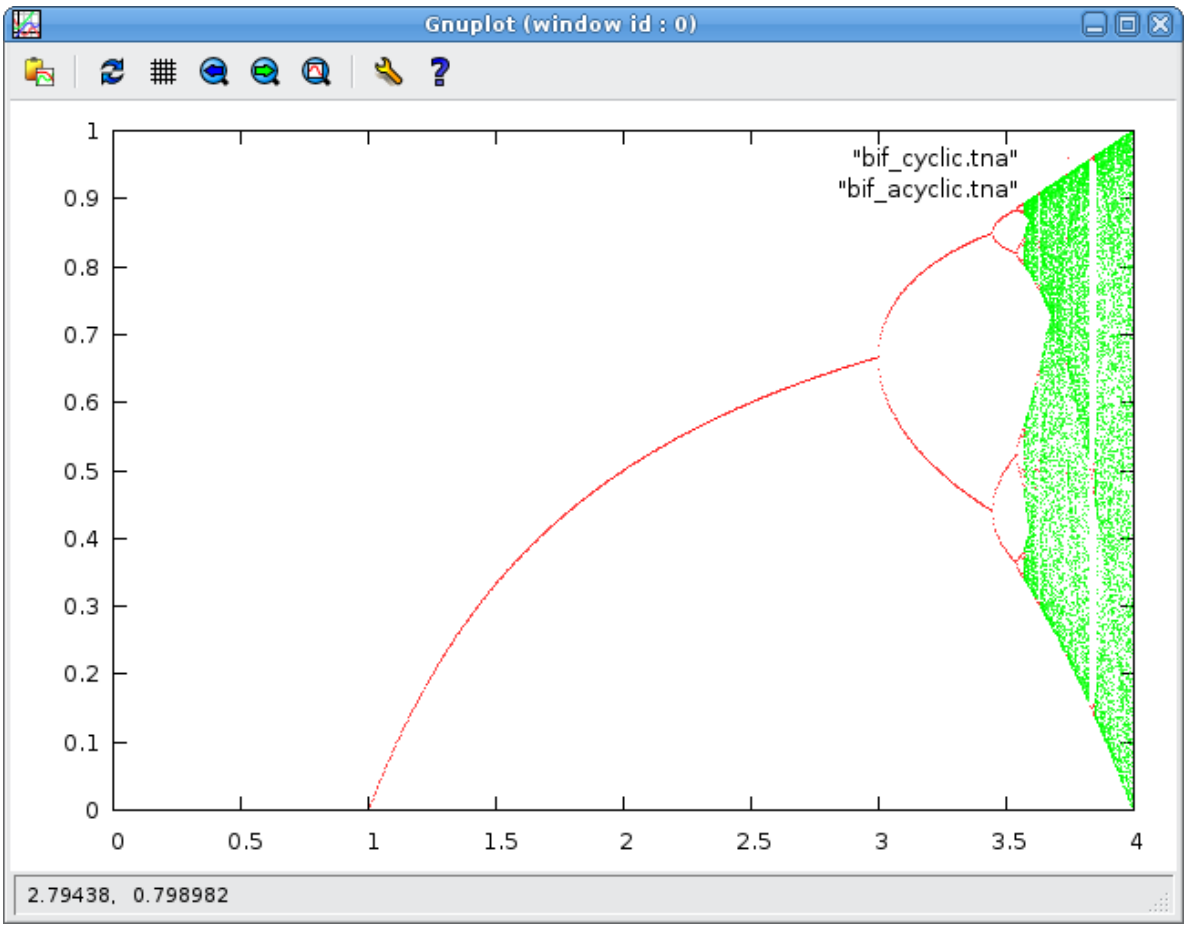

Figure 5.15: Bifurcation diagram

Note: `"with dots"` can be abbreviated as `"w d"`.

To display the period analysis type

```
gnuplot> plot 'period.tna' with dots
```

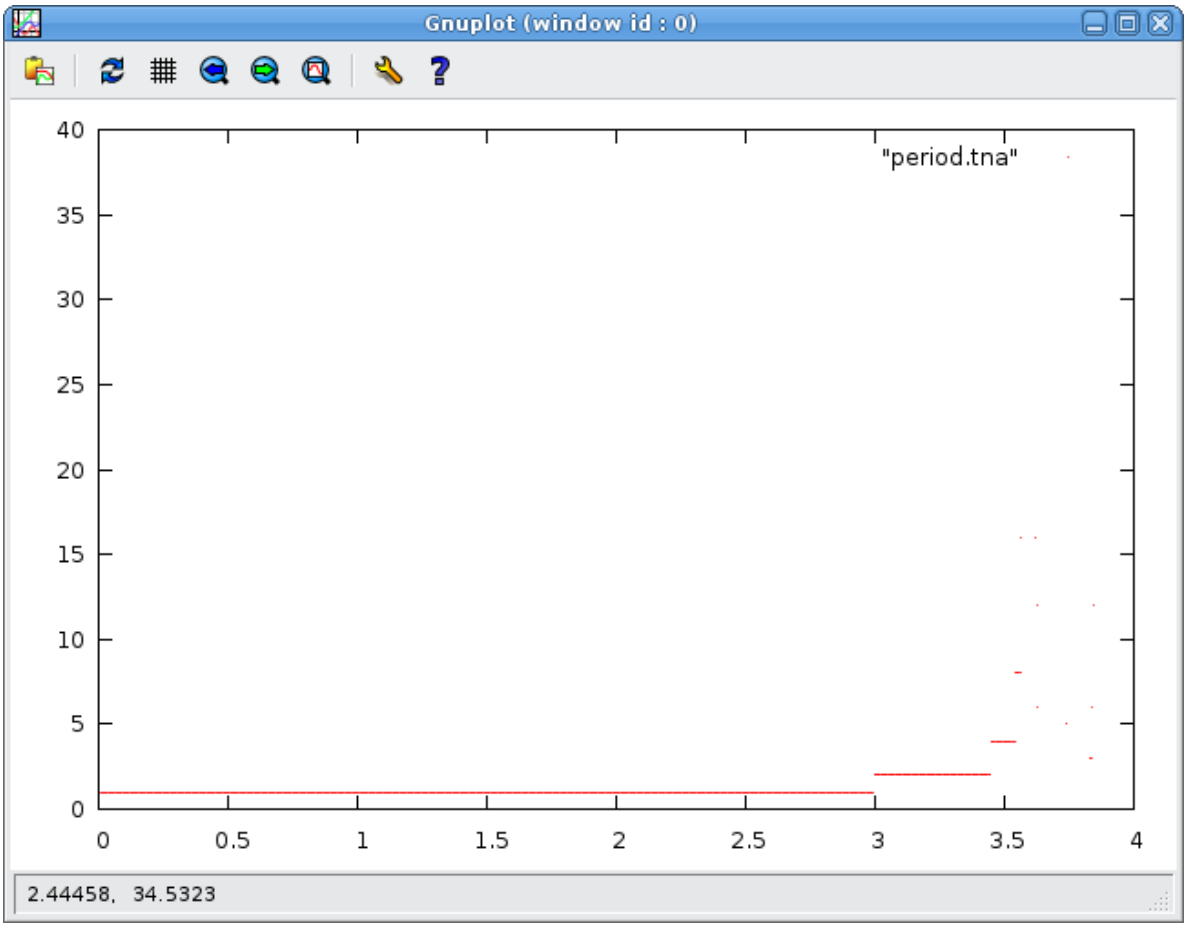

Figure 5.16: Period diagram

Lyapunov exponents greater than 0 are generally considered a sign of chaotic behavior. In order to visualize this threshold, we plot the graph of the function $f(x) = 0$ alongside the Lyapunov exponents by typing

```
gnuplot> plot 'lyapunov.tna' with dots, 0 notitle
```

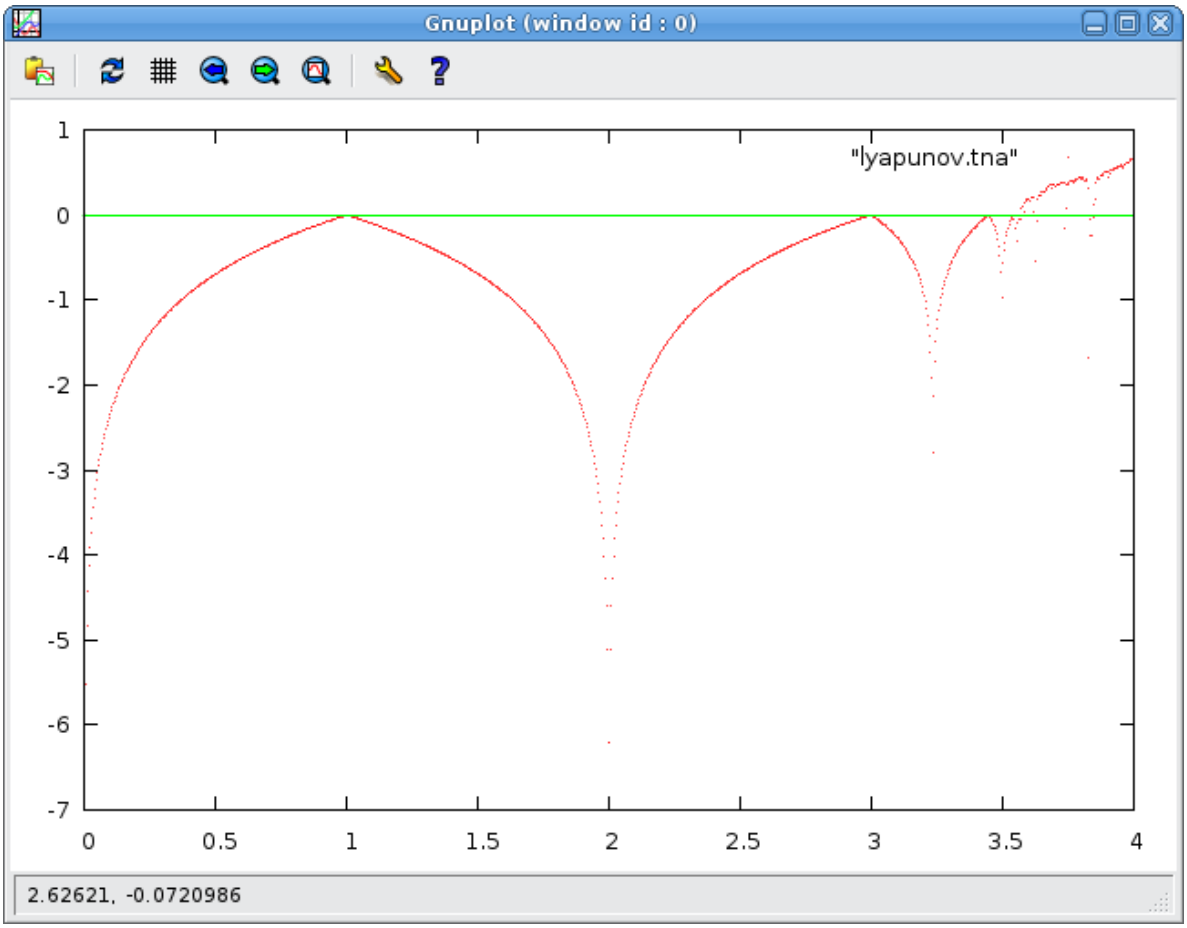

Figure 5.17: Lyapunov exponents

Note: The `"notitle"` option removes the function $f(x) = 0$ from the legend in the upper right hand corner of the gnuplot output.

Kapitel 6

Zusammenfassung und Ausblick

Das Problem der mangelnden Übersichtlichkeit und Bedienbarkeit der GUI von AnT 4.669 wurde gelöst, indem die zuvor vielen Fenster auf ein einziges reduziert wurden, in welchem die aktiven Konfigurationsknoten spaltenweise nebeneinander angeordnet sind.

Ausblick

Eine gerade für Einsteiger sehr verwirrende Eigenart der AnT 4.669-GUI ist die Tatsache, dass Änderungen in Konfigurationsfenstern erst übernommen werden, wenn selbiges geschlossen wird. Jeder Versuch, die Konfigurationsdatei vorher zu speichern, liefert die ursprüngliche, unveränderte Version. Um das erwartete Verhalten des „save“-Buttons (alle geänderten Optionen werden gespeichert) zu implementieren, müssten also vorher alle Daten von allen geöffneten Konfigurationsfenstern übernommen werden. Nun führt die AnT 4.669-GUI jedoch beim Schließen eines Fensters eine Fehlerprüfung durch, die nicht übersprungen werden sollte, um die Konsistenz der Konfigurationsdatei zu gewährleisten. Diese Fehlerprüfung könnte bei einer Speicherung nach der neuen Semantik nicht mehr in dieser Form ablaufen, da die Fenster ja nicht geschlossen werden sollen. Um dieses Problem ohne eine grundlegende Veränderung der Fehlerprüfung zu lösen, könnte man bei einem Speichervorgang beispielsweise das sequenzielle Schließen der Fenster simulieren und Fehler interaktiv vom Benutzer bearbeiten lassen. Leider liegen manche Fehler, wie zum Beispiel fehlende Angaben nicht auf dem aktiven Konfigurationspfad und der Benutzer müsste „abzweigen“, um sie zu beheben, was eine Wiederherstellung der Fensterhierarchie vor dem Speichervorgang erschwert.

Alle URLs wurden zuletzt am 23.10.2008 geprüft.

Erklärung

Hiermit versichere ich, diese Arbeit selbständig verfasst und nur die angegebenen Quellen benutzt zu haben.

(Andreas Wild)