Documentation for dynsys2.h and dynsys2.c

Steven Andrews, © 2008-2016

Header:

```
#ifndef __dynsys2_h__
#define __dynsys2_h__

typedef struct odestruct {
  int maxdim;              // allocated system dimensionality
  int dim;                 // actual system dimensionality
  int order;               // order of ODE integration (1=Euler)
  double *dtptr;           // pointer to dt
  double dtsugg;           // suggested time step for order 5
  double dtmax;            // maximum dt for order 5 integration
  double eps;              // maximum error for order 5 integration
  void *systemptr;         // pointer to system
  int (*eqm)(void *);      // function for equations of motion
  double **statenow;       // state at end of time step
  double **statewas;       // state at beginning of time step
  double **deriv;          // time derivative
  double *scale;           // scales errors for order 5 integration
  double *k1,*k2,*k3,*k4;  // scratch space for integrators
  } *odeptr;

odeptr dyns_AllocOde(int maxdim);
void dyns_FreeOde(odeptr ode);
int dyns_SetOrder(odeptr ode,int order);
int dyns_SetParamPtr(odeptr ode,char *param,void *value);
int dyns_SetParamDbl(odeptr ode,char *param,double value);
int dyns_AddStatePtr(odeptr ode,double *nowptr,double *wasptr,double scale);
void dyns_ClearStatePtrs(odeptr ode);
int dyns_StepOde(odeptr ode);

// Example program
int dyns_ShoExample(void);

#endif
```

History: First parts written 1/29/97. Field routines added 6/97; some testing. Slight additions 1/00. Reformatted documentation and added a new independent section 6/04, called '2004 version'. Fixed order 5 in 2004 version 9/1/04. Renamed to dynsys2 10/24/08, cut all pre-2004 stuff, and completely overhauled. Added `dyns_ClearStatePtrs` 4/9/09. Added `deriv` element to the data structure 4/4/14 and modified routines to account for it.

Data structure

`odestruct`, pointed to by a `odeptr`, is a structure that contains all the useful information about an ODE, including scratch space for the integrator. `maxdim` is the allocated system dimensionality (maximum number of differential equations). `dim` is the

actual system dimensionality. `order` is the order of the Runge-Kutta integrator, which is 1 for Euler integration, 2 for the mid-point method, 4 for 4th order Runge-Kutta, or 5 for 5th order Runge-Kutta with adaptive step-sizing. `dtptr` is a pointer to the variable that contains the time step. For order 5, `dtptr` is used to make `dt` equal to the actual size step that was taken, `dtsugg` is the suggested size of the next step, `dtmax` is the maximum time step allowed, and `eps` is the absolute maximum error allowed in any state variable. Of these, only `dtptr` is used for orders 1, 2, or 4. `systemptr` is a pointer to a structure that contains everything that is known about the system, cast as a `void*`; it is never used in any of the routines here, but it is passed on to the relevant function with the equations of motion. `eqm` is the function for the equations of motion, where its only parameter is the system, pointed to by `systemptr`. `statenow` and `statewas` are `maxdim` long lists of pointers, where each pointer points to one of the time dependent variables. `scale` is a list of values that are used to scale the calculated errors for step size control, used for order 5 integration. `k1` to `k4` are `dim` size vectors for scratch space and are only allocated as needed.

When the integrator is done with one time step, the variables pointed to by `statenow` contain the state of the system at the end of the time step, while those pointed to by `statewas` contain the state of the system at the beginning of the time step. While the integrator is busy, `statewas` points to the current system values and `deriv` is supposed to be returned to the integrator from the equations of motion function with the time derivative for each variable (it was statewas up to 4/4/14). This is made clearer below.

Using the routines

The routines are demonstrated in the example function called `dyns_ShoExample`, which can be called from externally. It simulates a simple harmonic oscillator with damping.

The `scale` vector can be useful for improving the quality of step sizing for order 5 integration. As a default, all `scale` values are equal to 1.0 to maintain an unscaled absolute error. Alternatively, values could be set to "typical" values for that state variable to still maintain absolute errors, but now without focusing on the smallest valued state variables. Also, `scale` could be reset at each time step to equal the absolute value of the state variable to maintain a constant relative error, but one needs to make sure that it is never zero.

Equations of motion

The integrator needs to have access to a function that contains the equations of motion. This function is given the system, cast as a void*, and needs to calculate the time derivatives of each state variable. Using the "`was`" and "`now`" notation that is used in this library, the equations of motion need to use the "`was`" state information to calculate time derivatives, which it stores in the "`deriv`" variables. This function should return 0 for correct operation and 1 for failure. Failure will cause integrator termination.

Functions

```
odeptr dyns_AllocOde(int maxdim);
```

Allocates an `odestruct` and returns an `odeptr` that points to it, assuming allocation was successful, or `NULL` if not. `maxdim` is the maximum number of differential equations (number of time dependent variables).

```
void dyns_FreeOde(odeptr ode);
```
Frees an `odestruct`, including all memory that it owns. It does not free memory that was allocated elsewhere.

```
int dyns_SetOrder(odeptr ode,int order);
```
Sets the integration order of an `odestruct`. This function can be called as often as wanted, and in any sequence. Allowable `order` values are 1, 2, 4, or 5. Returns 0 for success, 2 for illegal inputs, or 1 for inability to allocate memory.

```
int dyns_SetParamPtr(odeptr ode,char *param,void *value);
```
Sets pointer parameters of an `odestruct`, several of which must be set before the `odestruct` is used. The `param` string can be "`dtptr`", "`systemptr`", or "`eqm`"; in each case, the `value` should be the appropriate pointer, cast as a `void*` if necessary. Returns 0 for success or 2 for illegal inputs.

```
int dyns_SetParamDbl(odeptr ode,char *param,double value);
```
Sets floating point parameters of an `odestruct`. The `param` string can be "`dtsugg`", "`dtmax`", or "`eps`"; in each case, the `value` should be the appropriate number. Returns 0 for success or 2 for illegal inputs (including zero or negative values). This function only needs to be used for order 5 integration.

```
int dyns_AddStatePtr(odeptr ode,double *nowptr,double *wasptr,double
*derivptr,double scale);
```
Adds a state variable to the odestruct. Send in `nowptr` pointing to the variable that will contain the current state value, `wasptr` pointing to the variable that will contain the prior state value, and `derivptr` pointing to the variable that will contain the derivative information. `derivptr` may be the same as `nowptr`. `scale` is a value for the characteristic size scale of this variable; if it is sent in less than or equal to 0, the default value of 1 will be used.

```
void dyns_ClearStatePtrs(odeptr ode);
```
Clears all state variables from the ode but does not free any memory or change the `maxdim` element. After this is complete, the `dim` element equals zero.

```
int dyns_StepOde(odeptr ode);
```
Performs the integration of an `odestruct` over one time step. It uses Euler, Mid-point, fixed step Runge-Kutta, or adaptive step Runge-Kutta, depending on the `order` element. On input, `statenow` pointers point to the current state of the system and `statewas` pointers are ignored (i.e. there is no need to prepare the statewas values). On output, `statewas` pointers point to the past state of the system and `statenow` pointers point to the new state. Note that the derivative information is not set to zero before the equations of motion function is called. The `eqm` function for the equations of motion is supposed to return 0 for correct operation and any other value for failure; on failure, returns the same error code and otherwise returns 0.

```
int dyns_ShoExample(void);
```

Example of the use of the functions in this library. This can be called and will just plain run. See below.

Example program

To show how to use these functions, and to test them, a simple harmonic oscillator example is included. Because the entire system needs to be encapsulated in a data structure, a data structure is used here, which is called `shostate`. It is:

```
typedef struct shostate {
  double positionwas;
  double positionnow;
  double velocitywas;
  double velocitynow;
  double omega2;
  double gamma;
  double time;
  double dt;
  } *shostateptr;
```

The equations of motion are

$$\dot{x} = v$$
$$\dot{v} = -\omega^2 x - 2\gamma v$$

x is the position, v is the velocity, ω is the frequency, and γ is a damping term. These are implemented in the equations of motion function called `ShoEqm`. The exact theoretical results are:

$$x = Ae^{-\gamma t}\cos(\omega' t - \phi)$$
$$v = Ae^{-\gamma t}\left[-\omega'\sin(\omega' t - \phi) - \gamma\cos(\omega' t - \phi)\right]$$
$$\text{where } \omega' = \sqrt{\omega^2 - \gamma^2}$$

Using order 1 integration, results are way off, with an amplitude that does not decrease nearly fast enough. Results are better with order 2 and excellent with orders 4 or 5. With order 5, it is important to limit the time step to the natural time constant of the system, which is ω^{-1}. Because absolute errors are kept below a threshold, the relative errors become large when absolute values are small, unless a maximum time step is enforced.