Computational Physics (6810): Activities 8

Dick Furnstahl
Nuclear Theory Group
OSU Physics Department

February 25, 2017

Differential equation solving

Activities 7 Follow-ups

Activities 8 Stuff

Solving differential equations step-by-step

- Builds on numerical derivatives

$$\frac{dx}{dt} = f(x,t) \quad \text{e.g., } \frac{dx}{dt} = -ax \text{ or } \frac{dx}{dt} = -bt \text{ or } \ldots$$

  - instruction on how to change x from one t value to a nearby one (i.e., $t + \Delta t = t + h$)

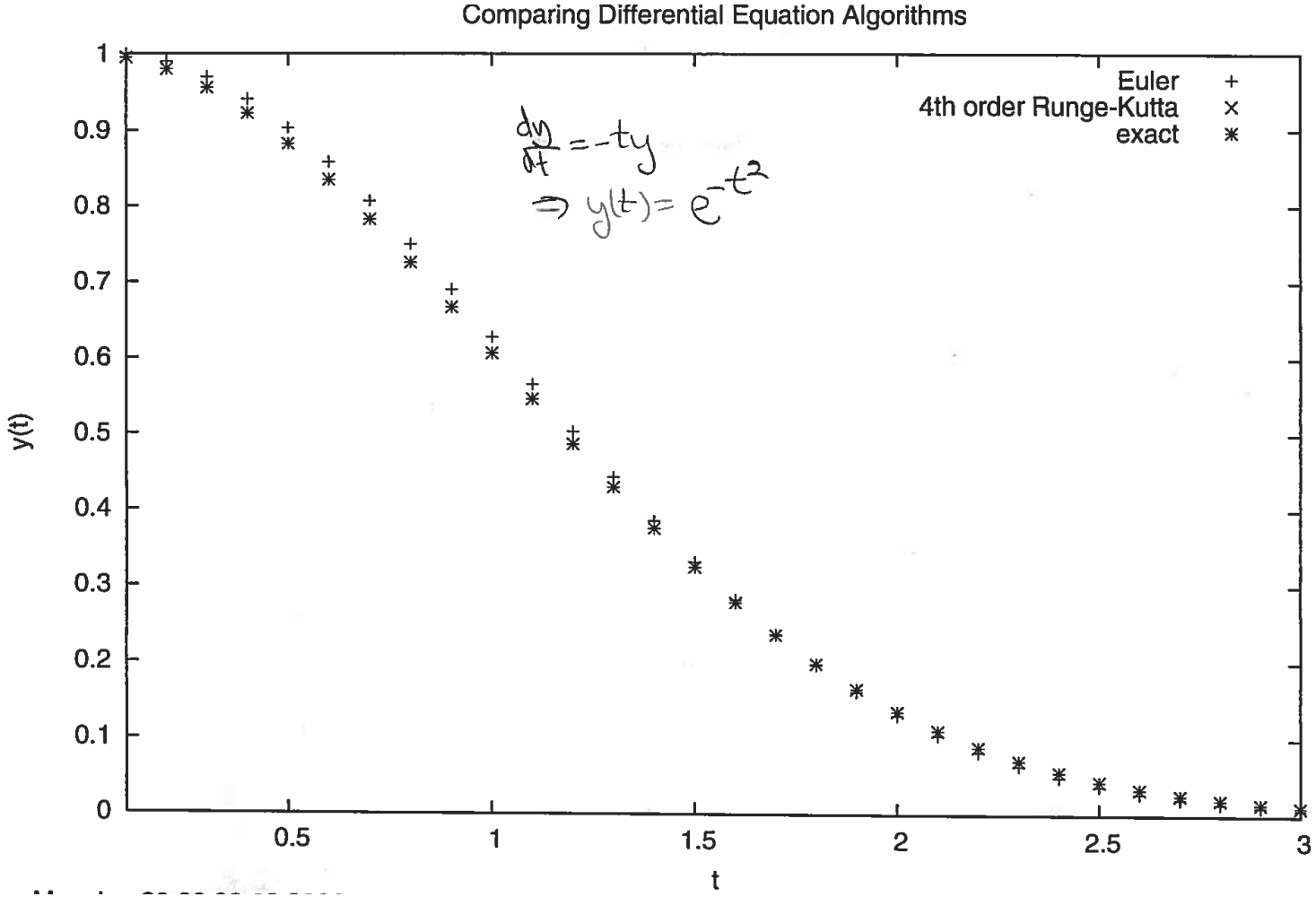
- Goal: Given $x(t = t_0)$, find $x(t = t_f)$
- Plan: the diff. eq. tells us how to take one step at a time
  $\Longrightarrow$ find $x(t_0 + h)$, then $x(t_0 + 2h)$, then $x(t_0 + 3h)$ until $x(t_f)$

$$x(t_0 + h) = x(t_0) + h \left.\frac{dx}{dt}\right|_{t=t_0} + \mathcal{O}(h^2) \text{ from Taylor expansion}$$

$$= x(t_0) + hf(x(t_0), t_0) + \mathcal{O}(h^2)$$

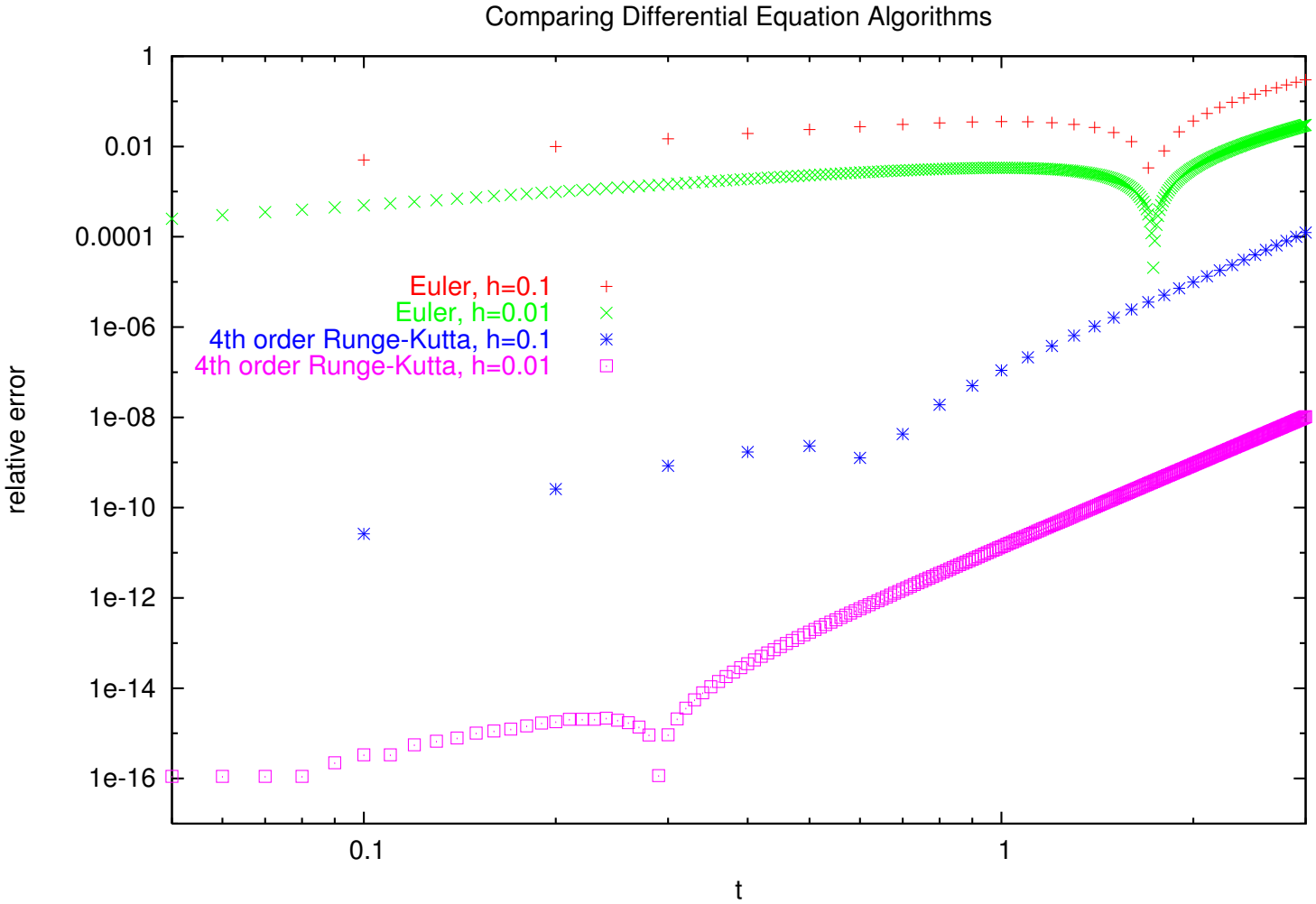
- How to do better?
  - Explore errors, other strategies, how to pick h, etc.
  - Some tricky parts: look for implementation errors if you get unexpected results

Differential equations graphs and their interpretation



- The differential equation *does not* have $y(t = 0)$ on the right side.
- The approximate solution crosses the exact sometimes. What happens to the error?

Differential equations graphs and their interpretation



- You should not interpret the slope as the diff. eq. error? Why not?
- How should you look for the global approximation error here?
- What is the local vs. global error? What are the dips from?

Activities 7: Implementation of coupled equations

- $y[0]$ is not y at time $t = 0$: it is the zeroth component of a vector
- Consider Newton's Second Law:

$$\frac{d^2x}{dt^2} = \frac{F(x, v, t)}{M} ,$$

re-express as vector differential equation:

$$\frac{d\mathbf{y}}{dt} = \mathbf{f} \quad \text{where} \quad \mathbf{y} = \begin{pmatrix} y^{(0)} \\ y^{(1)} \end{pmatrix} \quad \text{and} \quad \mathbf{f} = \begin{pmatrix} y^{(1)}(t) \\ \frac{1}{M} F(\mathbf{y}, t) \end{pmatrix}$$

if we make the definitions

$$y^{(0)}(t) \equiv x(t) \qquad \text{and} \qquad y^{(1)}(t) \equiv v = \frac{dx}{dt} = \frac{dy^{(0)}}{dt} .$$

- In the code, at each t

$$y^{(0)}(t) \longrightarrow \texttt{y[0]} \quad \text{and} \quad y^{(1)}(t) \longrightarrow \texttt{y[1]}$$

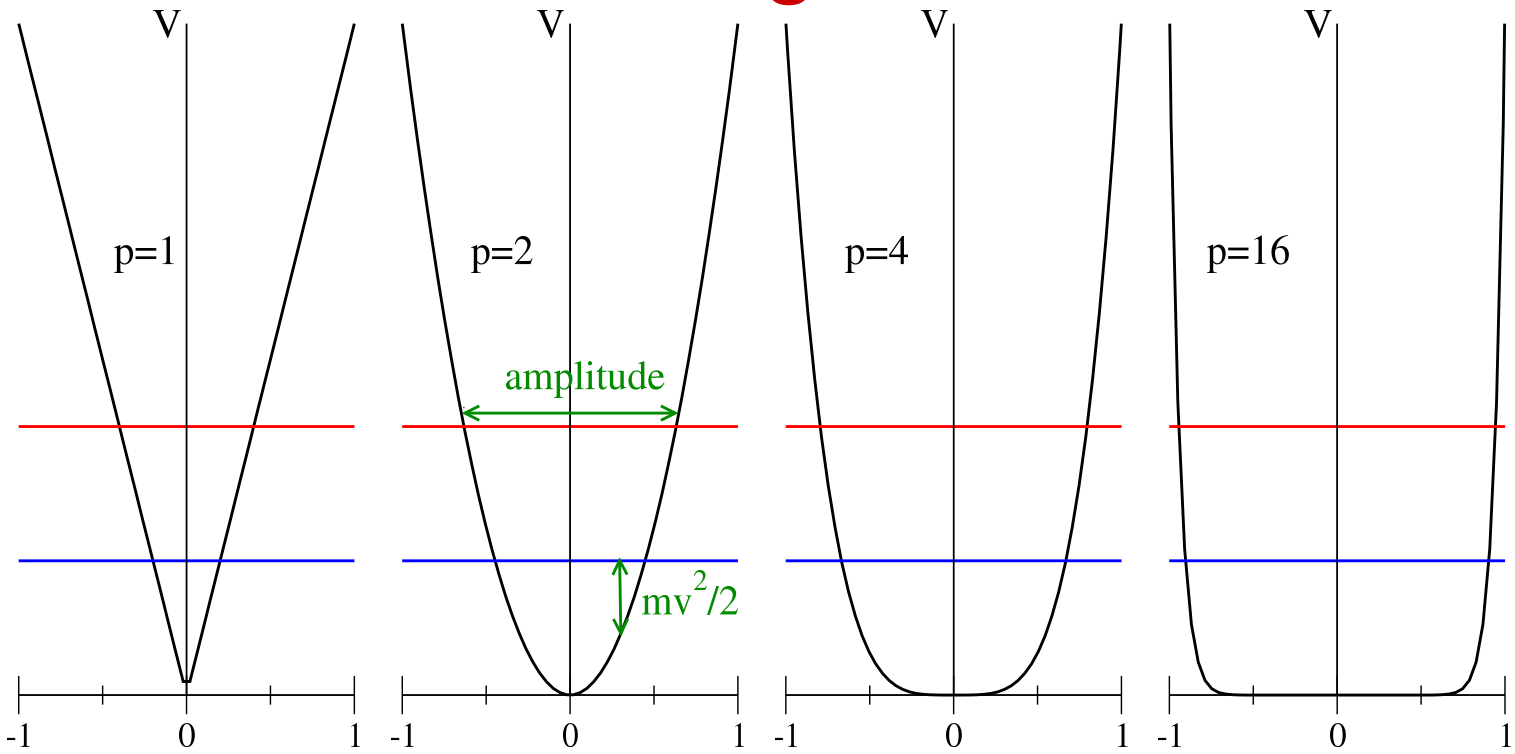
- Euler increment from t to $t + h$:

```
y[i] += h * f (t, y, i, params_ptr);
```

Activities 7: Other things to look for

- What step size h to use? See the Activities 7 notes for a discussion of how to make it *adaptive* (changes as you go to keep the error fixed).
  - Conserved quantites should be constant with time! Are they?
  - Remember to look at the *relative error*, not the full value
- Period vs. amplitude: for a harmonic oscillator they are independent. But not for other potentials!
  - Energy diagrams like those used for central potentials can help you understand the observed relation (see next slide)
- When doing damping: make sure phase space plots are reproduced (see handout)
  - With increasing time, does the phase space trajectory go clockwise, counter-clockwise, or both?

Activities 7: Other things to look for



- Where is the mass moving fastest and where slowest for each of the potential energies shown?
- To increase the amplitude, you need to increase the energy. What does this do to the speed?
- By how much does the amplitude increase in each case?
- Is the round-trip time (period) greater, smaller, the same?

Activities 8 Stuff

- Damped, driven harmonic oscillator

$$\ddot{x} + \gamma\dot{x} + \omega_0^2 x = f(t) \qquad \dot{x} \equiv dx/dt, \quad \ddot{x} \equiv d^2x/dt^2$$

  - *linear* equation $\Longrightarrow$ only single powers of $\ddot{x}$, $\dot{x}$, x
    $\Longrightarrow$ important for superposition:

$$x_{\text{total}}(t) = x_{\text{homogeneous}}(t) + x_{\text{particular}}(t)$$

  - $x_{\text{homogeneous}}(t)$ will always be damped $\Longrightarrow$ transient!
  - particular solution will have $x \propto e^{i\omega_{\text{ext}}t}$

$$\Longrightarrow (-\omega_{\text{ext}}^2 + i\omega_{\text{ext}}\gamma + \omega_0^2)\cancel{e^{-i\omega_{\text{ext}}t}} = A\cancel{e^{-i\omega_{\text{ext}}t}}$$

  - so the driving frequency ω_{ext} *is* the frequency in the linear domain $\Longrightarrow$ green dots are on top of each other
  - What if nonlinear? More interesting possibilities!

Chaos

- Characteristics of chaos (see Activities 8 notes)
  - past behavior not repeated (not periodic)
  - deterministic but not predictable, because uncertainty (or imprecision) in initial conditions grows exponentially in time
  - system has distributed power spectrum (see Mathematica notebooks)
- Necessary conditions for chaos
  - $\geq$ 3 independent variables and the equations have nonlinear terms coupling
  - Three equations for the pendulum (with $\phi = \omega_{\text{ext}} t$)

$$\frac{d\theta}{dt} = \omega \qquad \frac{d\omega}{dt} = -\alpha\omega \underbrace{- \omega_0^2 \sin\theta - f_{\text{ext}} \cos\phi}_{\text{nonlinear couplings}} \qquad \frac{d\phi}{dt} = \omega_{\text{ext}}$$

- Activities 10: Mathematica notebook `pendulum.nb`
  - gives results for Activities 8 “Looking for Chaos”
  - power spectrum: what frequencies are in the $\theta(t)$ plot?

GslHamiltonian class: Sample usage

```
// Create the Hamiltonian object called my_hamiltonian
Hamiltonian my_hamiltonian(dimension);

// Load the Hamiltonian matrix pointed to by Hmat_ptr
for (int i = 1; i <= dimension; i++)
{
    for (int j = 1; j <= dimension; j++)
    {
      ho_parameters.i = i-1;
      ho_parameters.j = j-1;
      // set the i,j element to Hij
      my_hamiltonian.set_element(i,j,Hij(ho_parameters));
    }
}

// Find eigenvalues and eigenvectors in ascending order
my_hamiltonian.find_eigenstuff();

// Get the first eigenvalue [i=1]
double eigenvalue = my_hamiltonian.get_eigenvalue(1);
```

GslHamiltonian class header

```
#include <gsl/gsl_eigen.h>  // include the GSL header file
class Hamiltonian
{
  public:
    Hamiltonian (const int dim);  // constructor [when declared]
    ~Hamiltonian ();  // destructor [when destroyed or at end]

    // accessor functions
    void set_element(const int i, const int j, double value);
    void find_eigenstuff();
    double get_eigenvalue(const int i);
    double get_eigenvector(const int i, const int j);
  private:
    int dimension;  // the matrix dimension
    gsl_matrix *Hmat_ptr;    // gsl matrix with Hamiltonian
    gsl_vector *Eigval_ptr;  // gsl vector with eigenvalues
    gsl_matrix *Eigvec_ptr;  // gsl matrix with eigenvectors
    gsl_eigen_symmv_workspace *worksp;  // the workspace for gsl
    gsl_vector *eigenvector_ptr;  // one of the eigenvectors
};
```

GslHamiltonian class constructor

```
// Constructor for Hamiltonian (same name as class!)
Hamiltonian::Hamiltonian (const int dim)
{
  dimension = dim;  // dimension is a private class variable

  // Allocate space for the vectors, matrices, and workspace
  // Hamiltonian
    Hmat_ptr = gsl_matrix_alloc (dimension, dimension);

  // eigenvalue vector
    Eigval_ptr = gsl_vector_alloc (dimension);
  // eigenvector matrix
    Eigvec_ptr = gsl_matrix_alloc (dimension, dimension);
  // one eigenvector
    eigenvector_ptr = gsl_vector_alloc (dimension);

  // workspace
    worksp= gsl_eigen_symmv_alloc (dimension);

}
```

GslHamiltonian class destructor and one method

```
Hamiltonian::~Hamiltonian () // Destructor for Hamiltonian
{
  // free the space used by the vectors, matrices, workspace
  gsl_matrix_free (Eigvec_ptr);
  gsl_vector_free (Eigval_ptr);
  gsl_matrix_free (Hmat_ptr);
  gsl_vector_free (eigenvector_ptr);
  gsl_eigen_symmv_free (worksp);
}


// Set an element [Note the use of "const" here!]
void Hamiltonian::set_element(const int i, const int j,
                              const double value)
{
  // The i,j element of the matrix is stored as the
  //     i-1,j-1 element of the GSL matrix
  gsl_matrix_set (Hmat_ptr, i-1, j-1, value);
}
```

GslHamiltonian class other methods

```
void Hamiltonian::find_eigenstuff()
{
  // Find the eigenvalues and eigenvectors of the real, symmetric
  //  matrix pointed to by Hmat_ptr.  [other comments suppressed]
  gsl_eigen_symmv (Hmat_ptr, Eigval_ptr, Eigvec_ptr, worksp);

  // Sort the eigenvalues and eigenvectors in ascending order
  gsl_eigen_symmv_sort (Eigval_ptr, Eigvec_ptr,
                        GSL_EIGEN_SORT_VAL_ASC);
}

double Hamiltonian::get_eigenvalue(int i)  // i'th eigenvalue
{
  return gsl_vector_get (Eigval_ptr, i-1);
}
double Hamiltonian::get_eigenvector(int i, int j)
{
  gsl_matrix_get_col (eigenvector_ptr, Eigvec_ptr, i-1);
  return gsl_vector_get (eigenvector_ptr, j-1);
}
```

How about a Potential class? What features would you like?

- Define different potentials by name
- Allow for changing parameters
- Be able to get the value at radius *r*

```
// Create a Potential object
V0 = -30.; R_width = 3.;   // depth and width
Potential my_square_well("Square well", V0, R_width);

// Create another Potential object (Z*e*e strength)
Potential my_coulomb_potential("Coulomb", Ze_sq)

// Evaluate the potentials
r = 2.;
V1 = my_square_well.evaluate(r);
V2 = my_coulomb_potential.evaluate(r);

// Change the square well width
R_width = 4.;
my_square_well.set_width(R_width);
```