Scientific Computing (Phys 2109/Ast 3100H)
II. Numerical Tools for Physical Scientists

SciNet HPC Consortium

University of Toronto

Winter 2013

Lecture 11

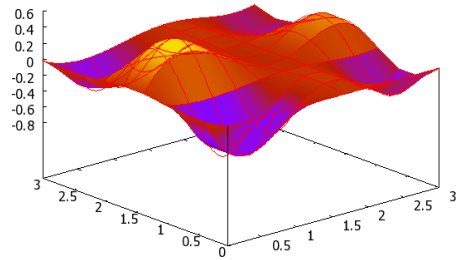
Numerical Integration

Solving ordinary differential equations

SciNet
compute • calcul
CANADA

Numerical Integration

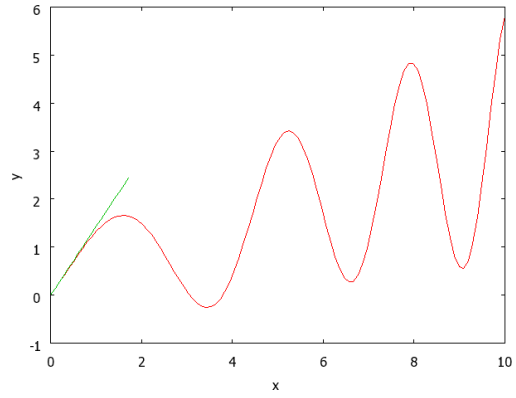
$$\mathcal{I} = \int_{\mathcal{D}} f(x)\, \mathrm{d}^d x$$



Ordinary Differential Equation

$$\sum_n a_n(x,y)\frac{\mathrm{d}^n y}{\mathrm{d}x^n} = f(x,y)$$

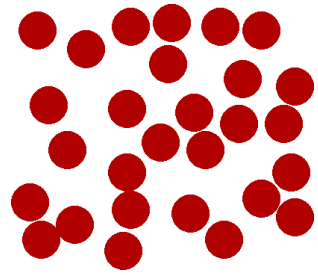
+ boundary/initial conditions



Molecular Dynamics Simulations

$$m_i \ddot{r}_i = f_i(\{r\},\{\dot{r}_j\},t)$$

+ initial conditions



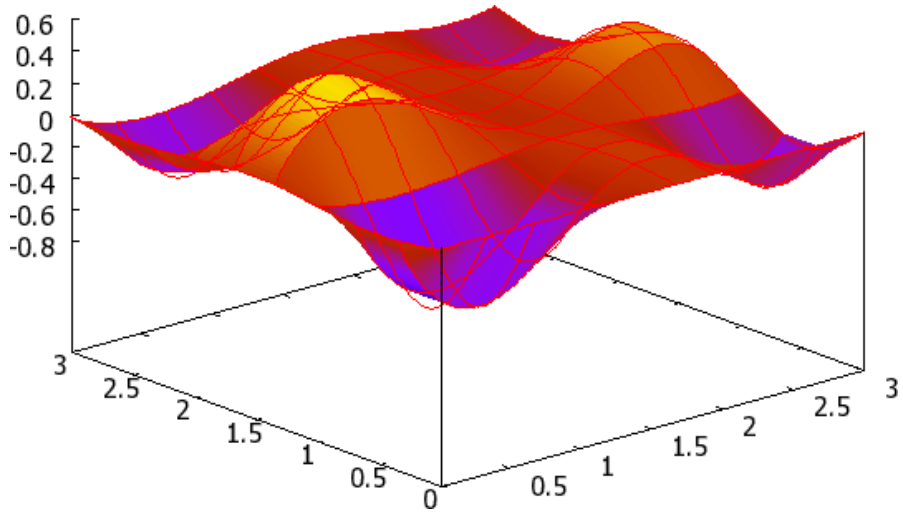
SciNet
compute • calcul
CANADA

Numerical Integration

Numerical Integration, or "Quadrature"

How to numerically estimate

$$\mathcal{I} = \int_{\mathcal{D}} f(x)\, \mathrm{d}^d x$$



Large variety of methods, depending on d, $f(x)$ and x

For $d = 1$:

$$\mathcal{I} = \int_a^b f(x)\, \mathrm{d}x$$

1. Regular grid
2. Gaussian Quadrature

Small d:

1. Regular grid
2. Recursive Quadrature

Large d:

1. Monte Carlo

Numerical Integration in $d = 1$

Regularly spaced grid method #1

$$\mathcal{I}(a,b) = \int_a^b f(x)\,\mathrm{d}x.$$

On small interval $[a, a+h]$, interpolate using values at a few points.

- Interpolating polynomial of degree 0 using mid-point:

$$\int_a^{a+h} f(x)\,\mathrm{d}x \approx h\, f\left(a + \frac{h}{2}\right)$$

- Linear interpolation based on end-points: Trapezoidal rule

$$\int_a^{a+h} f(x)\,\mathrm{d}x \approx \frac{h}{2}\left[f(a) + f(a+h)\right]$$

- Compose trapezoidal rule $n\times$ on sub-intervals $[kh, (k+1)h]$ $(k = 0, \ldots, n-1;\ h = (b-a)/n)$: Extended trapezoidal rule

$$\int_a^b f(x)\,\mathrm{d}x \approx h\left[\frac{f(a)+f(b)}{2} + \sum_{k=1}^{n-1} f(a+kh)\right] + \mathcal{O}\left(\frac{1}{n^2}\right)$$

Numerical Integration in $d = 1$

Regularly spaced grid method #2

- Interpolating function of degree 2 on $[a, a+2h]$ using end-points and mid-point:
  Simpsons' rule

$$\int_a^{a+2h} f(x)\,\mathrm{d}x \approx h\left[\tfrac{1}{3}f(a) + \tfrac{4}{3}f(a+\tfrac{h}{2}) + \tfrac{1}{3}f(a+h)\right]$$

- Compose n times on full interval:
  Extended Simpsons' rule

$$\int_a^b f(x)\,\mathrm{d}x \approx h\Big[\tfrac{1}{3}f(a) + \tfrac{4}{3}f(a+h) + \tfrac{2}{3}f(a+2h) + \tfrac{4}{3}f(a+3h) + \tfrac{2}{3}f(a+4h) + \cdots + \tfrac{1}{3}f(b)\Big] + \mathcal{O}\left(\frac{1}{n^4}\right)$$

SciNet compute • calcul CANADA

Numerical Integration in $d = 1$

Method using unevenly spaced grid: Gaussian quadrature

- Based on orthogonal polynomials on the interval.
  E.g. Legendre, Chebyshev, Hermite, Jacobi polynomials
- Compute and $f_i = f(x_i)$ then

$$\int_a^b f(x)\,\mathrm{d}x \approx \sum_{i=1}^{n} v_i f_i$$

  with choice of x_i and v_i based on zeroes of polynomial of degree n and of integrals of orthogonal polynomials.
- Well-defined procedure to find $\{x_i\}$ and $\{v_i\}$ (see e.g. *Numerical Recipes*).
- Error roughly the same as Simpsons' rule but as if $n \to 2n$.

Numerical Integration in $d = 1$

Specifiying accuracy

We may know the order of the error term, but not the accuracy.

$$\Downarrow$$

Good numerical integration routines increases n until some specified (relative or absolute) accuracy is achieved.

- Easier with fixed grid because old points get reused.
- But in standard Gaussian quadrature, the $\{x_i\}$ for n and for $n + 1$ have no points in common.
- Gauss-Kronrod methods allow reuse, but require specific sequences of n (e.g. 10, 21, 43, 87).

Numerical Integration in $d = 1$

Adaptive schemes

If a function is not smooth or behaves differently throughout the domains, divide and apply the above techniques to subdomains.

Weight functions

$$\mathcal{I} = \int_a^b w(x) f(x) \, \mathrm{d}x$$

There are ways to include weight w in the scheme.

- If w is standard, this can be done by changing the polynomials
- If w has singularities, this may remove numerical difficulties.

Don't code these yourself! Schemes like this, as well as Gaussian quadratures, are implemented in libraries such as the `gsl` and `boost::numeric::quadrature`.

GSL example (from the GSL docs)

```
#include <iostream>
#include <cmath>
#include <gsl/gsl_integration.h>
double f(double x,void *) { return log(x)/sqrt(x); }
int main() {
   int npts = 100;
   double nint, nerr, a=0, b=1, answ=-4;
   gsl_function func;
   gsl_integration_workspace* work;
   work = gsl_integration_workspace_alloc(npts);
   func.function = &f;
   gsl_integration_qags(&func, a, b, 0, 1e-7,
                        npts, work, &nint, &nerr);
   std::cout << "result          =" << nint << "\n"
             << "exact result    =" << answ << "\n"
             << "estimated error=" << nerr << "\n"
             << "actual error    =" << quad-answ << "\n"
             << "intervals       =" << work->size<< "\n";
   gsl_integration_workspace_free(work);
}
```

GSL documentation

www.gnu.org/software/gsl/manual/html_node/QAGS-adaptive-integration-with-singularities.html

17.4 QAGS adaptive integration with singularities

The presence of an integrable singularity in the integration region causes an adaptive routi
subintervals around the singularity. As the subintervals decrease in size the successive app
converge in a limiting fashion. This approach to the limit can be accelerated using an extra
QAGS algorithm combines adaptive bisection with the Wynn epsilon-algorithm to speed u
types of integrable singularities.

— Function: int **gsl_integration_qags** (*const gsl_function * f, double a, double b, double e*
*limit, gsl_integration_workspace * workspace, double * result, double * abserr*)

This function applies the Gauss-Kronrod 21-point integration rule adaptively until a
of f over (a,b) is achieved within the desired absolute and relative error limits, *epsab*
are extrapolated using the epsilon-algorithm, which accelerates the convergence of t
presence of discontinuities and integrable singularities. The function returns the fina
the extrapolation, *result*, and an estimate of the absolute error, *abserr*. The subinterv
stored in the memory provided by *workspace*. The maximum number of subintervals
may not exceed the allocated size of the workspace.

Numerical Integration in $d > 1$ but small.

Why multidimensional integration is hard:

- Requires $\mathcal{O}(n^d)$ points if its 1d counterpart requires n.
- A function can be peaked, and peak can easily be missed.
- The domain itself can be complicated.

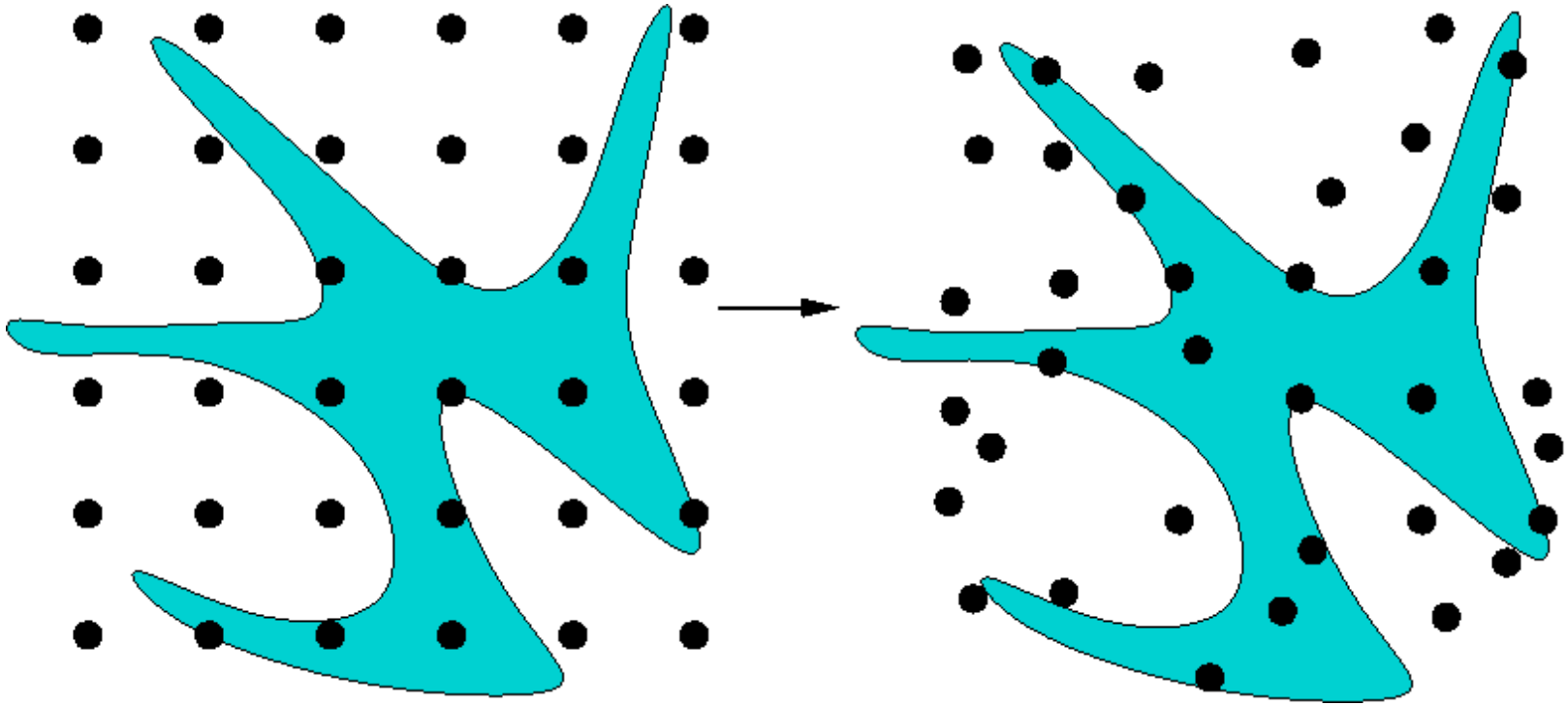
Numerical Integration in $d > 1$ but small.

So what should you do?

- If you can reduce the d by exploiting symmetry or doing part of the integral analytically, do it!
- If you know the function to integrate is smooth and its domain is fairly simple, you could do repeated 1d integrals (fixed-grid or Gaussian quadrature)
- Otherwise, you'll have to consider Monte Carlo.

Monte Carlo Integration

Use random numbers to pick points at which to evaluate integrand.



Similar to the rejection/acceptance scheme of the previous lecture.

- Convergence always as $1/\sqrt{n}$, regardless of d.
- Simple and flexible.
- Can generalize to focus on important parts.

Importance Sampling

$$\mathcal{I} = \int_V f(x)\,\mathrm{d}x$$

Suppose $f(x)$ is non-zero only in specific x regions.

- Want to place more points in region where integrand is large.
- Define function $w(x)$ that tells which regions are significant.
  - Require $w(x) > 0$ for any point x in volume where $f \neq 0$.
  - Re-express integral as:

$$\mathcal{I} = \int_V \frac{f(x)}{w(x)}\, w(x)\,\mathrm{d}x$$

- Draw a set of n points $\{x_1, \ldots, x_n\}$ weighted by $w(x)$, then

$$\overline{I} \approx \frac{1}{n} \sum_{i=1}^{n} \frac{f(x_i)}{w(x_i)}$$

- Converges to right answer for $n \to \infty$ as $1/\sqrt{n}$.

How does this improve the rate of convergence?

- The statistical uncertainty is related to the variance σ_I^2 of $\bar{I}$:

$$\sigma_{\bar{I}}^2 = \frac{1}{n}\sum_i^n \langle \Delta I_i \Delta I_i \rangle \qquad \text{where} \qquad \Delta I_i = \frac{f(x_i)}{w(x_i)} - \bar{I}$$

  (assuming ΔI_i are statistically independent).
- Vastly different values of $f(x_i)/w(x_i)$ lead to large uncertainty.
- If $\alpha\, w(x_i) = f(x_i)$, then $f(x_i)/w(x_i) = \alpha$ and

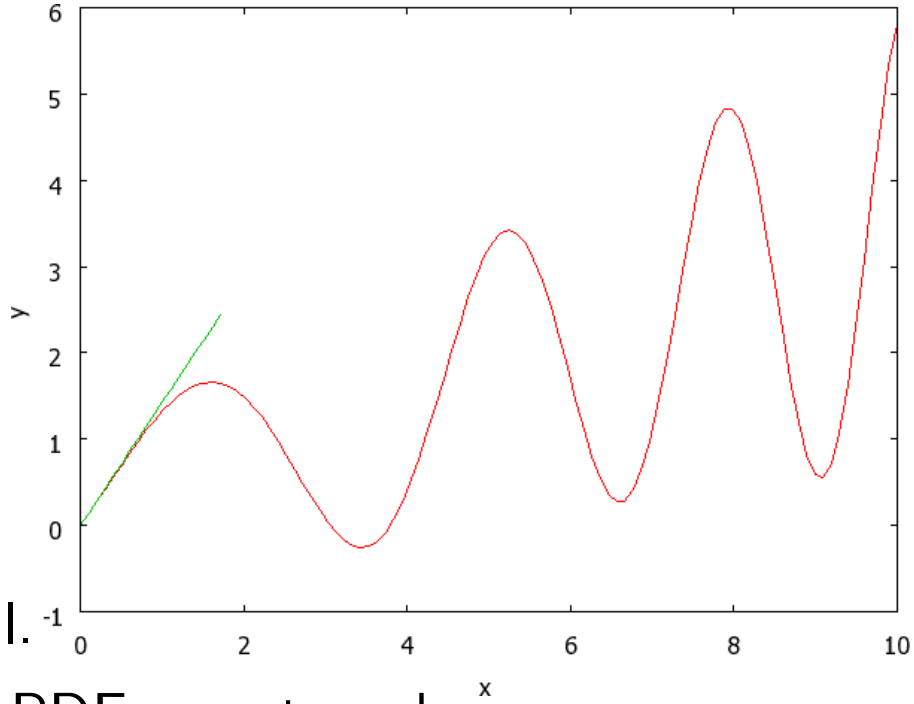
$$\left\langle \frac{f(x_i)}{w(x_i)} \right\rangle = I = \alpha \qquad \left\langle \left(\frac{f(x_i)}{w(x_i)} \right)^2 \right\rangle = \alpha^2,$$

  and $\sigma_{\bar{I}}^2 = 0$.
- Generally desire all $f(x_i)/w(x_i)$ to be roughly the same for all sampled points x_i to mimimize $\sigma_{\bar{I}}^2$.

ODE solvers

Ordinary Differential Equations (ODEs)

$$\sum_n a_n(x,y)\frac{\mathrm{d}^n y}{\mathrm{d}x^n} = f(x,y)$$



Example

$$\frac{\mathrm{d}^2 y}{\mathrm{d}x^2} = -y$$

- Ordinary $\rightarrow$ x is one dimensional.
- Boundary conditions: much like PDEs: next week.
- Initial conditions: y, $\frac{\mathrm{d}y}{\mathrm{d}x}$, ..., at $x = x_0$
- Define $y_0 = y$; $y_1 = \frac{\mathrm{d}y}{\mathrm{d}x}$, ..., $\rightarrow$ set of first order ODEs

  Example:

$$\frac{\mathrm{d}y_0}{\mathrm{d}x} = y_1$$

$$\frac{\mathrm{d}y_1}{\mathrm{d}x} = -y_0$$

SciNet compute • calcul CANADA

Numerical approaches

Start from the general form:

$$\frac{\mathrm{d}y_i}{\mathrm{d}x} = f(x, \{y_j\})$$

- All approaches will evaluate f at discrete points x_0, x_1, ...
- Initial conditions: specify $y_i(x_0)$ and $\frac{\mathrm{d}y_i}{\mathrm{d}x}(x_0)$.
- Consecutive points may have a fixed step size $h = x_{k+1} - x_k$ or may be adaptive.
- $\{y_j(x_{i+1})\}$ may be implicitly dependent on f at that value.

Stiff ODEs

- A stiff ODE is one that is hard to solve, i.e. requiring a very small stepsize h or leading to instabilities in some algoritms.
- Usually due to wide variation of time scales in the ODEs.
- Not all methods equally suited for stiff ODEs

ODE solver algorithms: Euler

To solve:

$$\frac{dy}{dx} = f(x, y)$$

Simple approximation:

$$y_{n+1} \approx y_n + hf(x_n, y_n) \qquad \text{“forward Euler”}$$

Rational:

$$y(x_n + h) = y(x_n) + h\frac{dy}{dx}(x_n) + \mathcal{O}(h^2)$$

So:

$$y(x_n + h) = y(x_n) + hf(x_n, y_n) + \mathcal{O}(h^2)$$

- $\mathcal{O}(h^2)$ is the local error.
- For given interval $[x_1, x_2]$, there are $n = (x_2 - x_1)/h$ steps
- Global error: $n \times \mathcal{O}(h^2) = \mathcal{O}(h)$
- Not very accurate, nor very stable (next): don't use.

Stability

Example: solve harmonic oscillator numerically:

$$\frac{\mathrm{d}y^{(1)}}{\mathrm{d}x} = y^{(2)}$$

$$\frac{\mathrm{d}y^{(2)}}{\mathrm{d}x} = -y^{(1)}$$

Use Euler ($y_{n+1} \approx y_n + hf(x_n, y_n)$) gives

$$\begin{pmatrix} y_{n+1}^{(1)} \\ y_{n+1}^{(2)} \end{pmatrix} = \begin{pmatrix} 1 & h \\ -h & 1 \end{pmatrix} \begin{pmatrix} y_n^{(1)} \\ y_n^{(2)} \end{pmatrix}$$

Stability governed by eigenvalues $\lambda_\pm = 1 \pm ih$ of that matrix.
$|\lambda_\pm| = \sqrt{1+h^2} > 1 \quad \Rightarrow$ Unstable for any h!

ODE solver algorithms: implicit mid-point Euler

To solve:

$$\frac{\mathrm{d}y}{\mathrm{d}x} = f(x, y)$$

Symmetric simple approximation:

$$y_{n+1} \approx y_n + hf(x_n, (y_n + y_{n+1})/2) \qquad \text{“mid-point Euler”}$$

This is an implicit formula, i.e., has to be solved for y_{n+1}.

Example (Harmonic oscillator)

$$\begin{bmatrix} 1 & -\frac{h}{2} \\ \frac{h}{2} & 1 \end{bmatrix} \begin{bmatrix} y_{n+1}^{[1]} \\ y_{n+1}^{[2]} \end{bmatrix} = \begin{bmatrix} 1 & \frac{h}{2} \\ -\frac{h}{2} & 1 \end{bmatrix} \begin{bmatrix} y_n^{[1]} \\ y_n^{[2]} \end{bmatrix} \Rightarrow \begin{bmatrix} y_{n+1}^{[1]} \\ y_{n+1}^{[2]} \end{bmatrix} = M \begin{bmatrix} y_n^{[1]} \\ y_n^{[2]} \end{bmatrix}$$

Eigenvalues M are $\lambda_\pm = \frac{(1 \pm ih/2)^2}{1+h^2/4}$ so $|\lambda_\pm| = 1 \Rightarrow$ Stable for all h

Implicit methods often more stable and allow larger step size h.

ODE solver algorithms: Predictor-Corrector

1. Computation of new point
2. Correction using that new point

- Gear P.C.: keep previous values of y to do higher order Taylor series (predictor), then use f in last point to correct. Can suffer from catestrophic cancellation at very low h.
- Runge-Kutta: Refines by using mid-points. Workhorse even behind fancier solvers.

4th order version:

$$
\begin{aligned}
k_1 &= hf(x, y) \\
k_2 &= hf(x + h/2, y + k_1/2) \\
k_3 &= hf(x + h/2, y + k_2/2) \\
k_4 &= hf(x + h, y + k_3) \\
y' &= y + \frac{k_1}{6} + \frac{k_2}{3} + \frac{k_3}{3} + \frac{k_4}{6}
\end{aligned}
$$

Further ODE solver techniques

Adaptive methods

As with the integration, rather than taking a fixed h, vary h such that the solution has a certain accuracy.

Don't code this yourself! Adaptive schemes are implemented in libraries such as the `gsl` and `boost::numeric::odeint`.

Geometric, symplectic and variants

Respects hamiltonian form, better energy conservation.
Will discuss in the context of MD.