L11: Numerical Differentiation (Chapter 21)

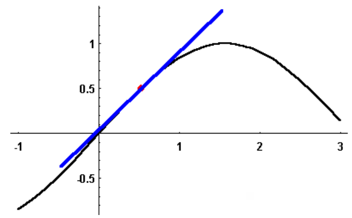
BME 313L
Introduction to Numerical Methods in BME

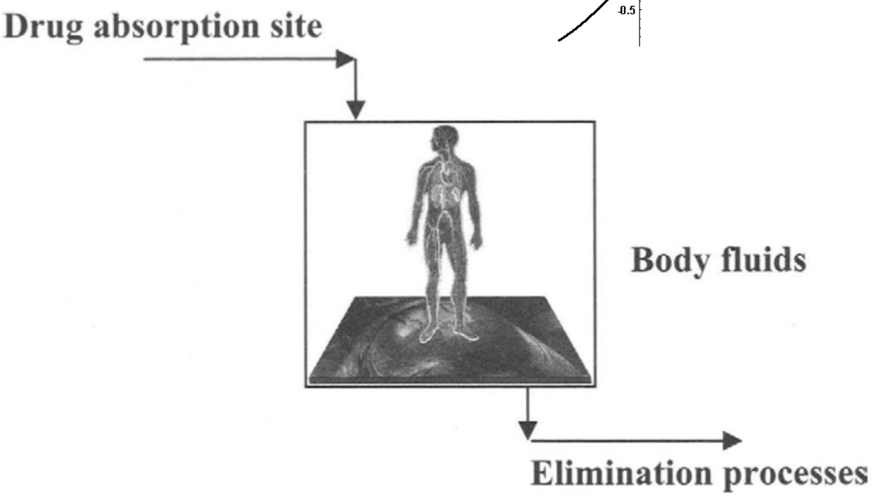
Tim Yeh
Department of Biomedical Engineering
University of Texas at Austin

$$\nabla \cdot \sigma_i \nabla \Phi_i = \beta C_m \frac{\partial V_m}{\partial t} + I_{ion}(V_m, q)$$

$$\nabla \cdot \sigma_e \nabla \Phi_e = -\beta C_m \frac{\partial V_m}{\partial t} - I_{ion}(V_m, q)$$

$$\frac{dq}{dt} = M(V_m q)$$





$$\frac{dP_n}{dt} = \frac{P_{n+1} - P_n}{C_n P_{n+1}} + \frac{P_{n-1} - P_n}{C_n R_n} + \frac{P_{bias} - P_n}{C_n} \cdot \left(\frac{dC_n}{dt}\right) + \frac{dP_{bias}}{dt}$$

Numerical Differentiation

- Differentiation formulas
  - Forward, backward and centered finite-difference
- Richardson extrapolation
- Unequally spaced data
- Data with errors
- Partial derivatives
- Differentiation with MATLAB
  - *diff* and *gradient*

Learning Objectives

- Understanding the application of **high-accuracy numerical differentiation** formulas for equispaced data.
- Knowing how to evaluate **derivatives for unequally spaced data**.
- Understanding how **Richardson extrapolation** is applied for numerical differentiation.
- Recognizing the **sensitivity** of numerical differentiation to data error.
- Knowing how to evaluate derivatives in MATLAB with the ***`diff`*** and ***`gradient`*** functions.
- Knowing how to generate **contour plots** and **vector fields** with MATLAB.

Differentiation

- The mathematical definition of a derivative begins with a **finite-difference approximation (Chapter 4, p.111)**:

$$\frac{\Delta y}{\Delta x}=\frac{f(x_i+\Delta x)-f(x_i)}{\Delta x}$$

- And as Δx is allowed to **approach zero**, the difference becomes a derivative:

$$\frac{dy}{dx}=\lim_{\Delta x\to 0}\frac{f(x_i+\Delta x)-f(x_i)}{\Delta x}$$

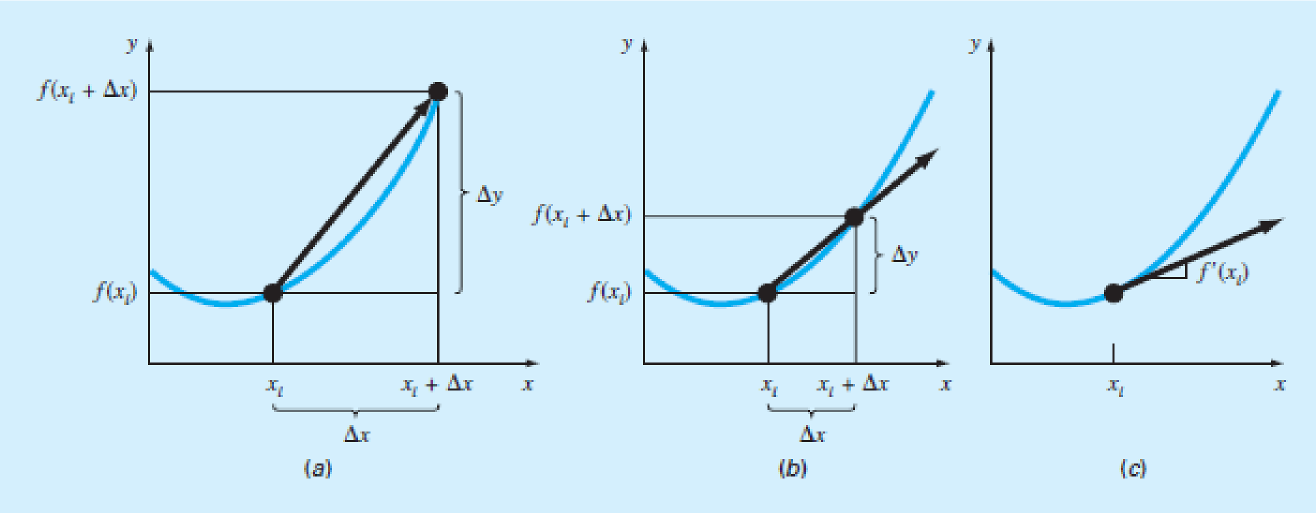


FIGURE 21.1
The graphical definition of a derivative: as Δx approaches zero in going from (a) to (c), the difference approximation becomes a derivative.

Differentiation

- What about 2nd derivative? The derivative of the 1st derivative:

$$\frac{d^2y}{dx^2} = \frac{d}{dx}\left(\frac{dy}{dx}\right)$$

- 2nd derivative is "how fast the slope is changing" – referred to as "curvature".
- Partial derivatives can be thought of as taking derivative of a function at a point with **all but one variable held constant**.

$$\frac{\partial f}{\partial x} = \lim_{\Delta x \to 0} \frac{f(x + \Delta x, y) - f(x, y)}{\Delta x} \qquad \frac{\partial f}{\partial y} = \lim_{\Delta y \to 0} \frac{f(x, y + \Delta y) - f(x, y)}{\Delta y}$$

The meaning of these partial derivatives?

The **mountain climbing** example – the **slope** to the **east ($\frac{\partial f}{\partial x}$)** and the **slope** to the **north ($\frac{\partial f}{\partial y}$)**.

Differentiation in Engineering

TABLE 21.1 The one-dimensional forms of some constitutive laws commonly used in engineering and science.

Law	Equation	Physical Area	Gradient	Flux	Proportionality
Fourier's law	$q = -k\dfrac{dT}{dx}$	Heat conduction	Temperature	Heat flux	Thermal Conductivity
Fick's law	$J = -D\dfrac{dc}{dx}$	Mass diffusion	Concentration	Mass flux	Diffusivity
Darcy's law	$q = -k\dfrac{dh}{dx}$	Flow through porous media	Head	Flow flux	Hydraulic Conductivity
Ohm's law	$J = -\sigma\dfrac{dV}{dx}$	Current flow	Voltage	Current flux	Electrical Conductivity
Newton's viscosity law	$\tau = \mu\dfrac{du}{dx}$	Fluids	Velocity	Shear Stress	Dynamic Viscosity
Hooke's law	$\sigma = E\dfrac{\Delta L}{L}$	Elasticity	Deformation	Stress	Young's Modulus

2^{nd} derivative?

Low-Accuracy Numerical Differentiation Formulas (Chap 4, p110)

- Forward-difference approximation for **1st derivative**:

$$f(x_{i+1}) = f(x_i) + f'(x_i)h + \frac{f''(x_i)}{2!}h^2 + \cdots$$

Eqn 4.21 $$f'(x_i) = \frac{f(x_{i+1}) - f(x_i)}{h} + \boxed{O(h)}$$

Not good enough

Finite-difference approximation for **2nd derivative**:

Eqn 4.24 $$f(x_{i+1}) = f(x_i) + f'(x_i)h + \frac{f''(x_i)}{2!}h^2 + \cdots$$

x (-2)

+

Eqn 4.26 $$f(x_{i+2}) = f(x_i) + f'(x_i)(2h) + \frac{f''(x_i)}{2!}(2h)^2 + \cdots$$

$$f(x_{i+2}) - 2f(x_{i+1}) = -f(x_i) + f''(x_i)h^2 + \cdots$$

Rearrange

Eqn 4.27 $$\boxed{f''(x_i) = \frac{f(x_{i+2}) - 2f(x_{i+1}) + f(x_i)}{h^2} + O(h)}$$

Low-Accuracy Numerical Differentiation Formulas

- Finite-difference approximation for **3rd derivative**:

$$f'''(x_i) = \frac{f(x_{i+3}) - 3f(x_{i+2}) + 3f(x_{i+1}) - f(x_i)}{h^3}$$

- Finite-difference approximation for **4th derivative**:

$$f''''(x_i) = \frac{f(x_{i+4}) - 4f(x_{i+3}) + 6f(x_{i+2}) - 4f(x_{i+1}) + f(x_i)}{h^4}$$

High-Accuracy Numerical Differentiation Formulas (Chap 21)

- In Chapter 21 (forward-difference formula):

Eqn 4.27 $$f''(x_i) = \frac{f(x_{i+2}) - 2f(x_{i+1}) + f(x_i)}{h^2} + O(h)$$

Plug in

Eqn 21.13 $$f'(x_i) = \frac{f(x_{i+1}) - f(x_i)}{h} - \frac{f''(x_i)}{2!}h + O(h^2)$$

Eqn 21.16
$$f'(x_i) = \frac{f(x_{i+1}) - f(x_i)}{h} - \frac{f(x_{i+2}) - 2f(x_{i+1}) + f(x_i)}{2h^2}h + O(h^2)$$

Rearrange

Eqn 21.17 $$f'(x_i) = \frac{-f(x_{i+2}) + 4f(x_{i+1}) - 3f(x_i)}{2h} + O(h^2)$$

- First derivative can be approximated by **a linear combination of 3 points** with reduced error → $O(h^2)$

Forward Difference Formulas (p. 527)

First Derivative | Error

$$f'(x_i) = \frac{f(x_{i+1}) - f(x_i)}{h} \qquad O(h)$$

$$f'(x_i) = \frac{-f(x_{i+2}) + 4f(x_{i+1}) - 3f(x_i)}{2h} \qquad O(h^2)$$

- Inclusion of the **2nd derivative** term
- As a result, using **3 points** for approximation

Second Derivative

$$f''(x_i) = \frac{f(x_{i+2}) - 2f(x_{i+1}) + f(x_i)}{h^2} \qquad O(h)$$

$$f''(x_i) = \frac{-f(x_{i+3}) + 4f(x_{i+2}) - 5f(x_{i+1}) + 2f(x_i)}{h^2} \qquad O(h^2)$$

Third Derivative

$$f'''(x_i) = \frac{f(x_{i+3}) - 3f(x_{i+2}) + 3f(x_{i+1}) - f(x_i)}{h^3} \qquad O(h)$$

$$f'''(x_i) = \frac{-3f(x_{i+4}) + 14f(x_{i+3}) - 24f(x_{i+2}) + 18f(x_{i+1}) - 5f(x_i)}{2h^3} \qquad O(h^2)$$

Fourth Derivative

$$f''''(x_i) = \frac{f(x_{i+4}) - 4f(x_{i+3}) + 6f(x_{i+2}) - 4f(x_{i+1}) + f(x_i)}{h^4} \qquad O(h)$$

$$f''''(x_i) = \frac{-2f(x_{i+5}) + 11f(x_{i+4}) - 24f(x_{i+3}) + 26f(x_{i+2}) - 14f(x_{i+1}) + 3f(x_i)}{h^4} \qquad O(h^2)$$

More terms of the Taylor series expansion are incorporated, **more accurate!**

Backward Difference Formulas (p. 528)

First Derivative	Error
$f'(x_i) = \dfrac{f(x_i) - f(x_{i-1})}{h}$	$O(h)$
$f'(x_i) = \dfrac{3f(x_i) - 4f(x_{i-1}) + f(x_{i-2})}{2h}$	$O(h^2)$
Second Derivative	
$f''(x_i) = \dfrac{f(x_i) - 2f(x_{i-1}) + f(x_{i-2})}{h^2}$	$O(h)$
$f''(x_i) = \dfrac{2f(x_i) - 5f(x_{i-1}) + 4f(x_{i-2}) - f(x_{i-3})}{h^2}$	$O(h^2)$
Third Derivative	
$f'''(x_i) = \dfrac{f(x_i) - 3f(x_{i-1}) + 3f(x_{i-2}) - f(x_{i-3})}{h^3}$	$O(h)$
$f'''(x_i) = \dfrac{5f(x_i) - 18f(x_{i-1}) + 24f(x_{i-2}) - 14f(x_{i-3}) + 3f(x_{i-4})}{2h^3}$	$O(h^2)$
Fourth Derivative	
$f''''(x_i) = \dfrac{f(x_i) - 4f(x_{i-1}) + 6f(x_{i-2}) - 4f(x_{i-3}) + f(x_{i-4})}{h^4}$	$O(h)$
$f''''(x_i) = \dfrac{3f(x_i) - 14f(x_{i-1}) + 26f(x_{i-2}) - 24f(x_{i-3}) + 11f(x_{i-4}) - 2f(x_{i-5})}{h^4}$	$O(h^2)$

Centered Difference Formulas (p.529)

Better than forward/backward formulas
(see p. 112)

	Error
First Derivative	
$f'(x_i) = \dfrac{f(x_{i+1}) - f(x_{i-1})}{2h}$	$O(h^2)$
$f'(x_i) = \dfrac{-f(x_{i+2}) + 8f(x_{i+1}) - 8f(x_{i-1}) + f(x_{i-2})}{12h}$	$O(h^4)$
Second Derivative	
$f''(x_i) = \dfrac{f(x_{i+1}) - 2f(x_i) + f(x_{i-1})}{h^2}$	$O(h^2)$
$f''(x_i) = \dfrac{-f(x_{i+2}) + 16f(x_{i+1}) - 30f(x_i) + 16f(x_{i-1}) - f(x_{i-2})}{12h^2}$	$O(h^4)$
Third Derivative	
$f'''(x_i) = \dfrac{f(x_{i+2}) - 2f(x_{i+1}) + 2f(x_{i-1}) - f(x_{i-2})}{2h^3}$	$O(h^2)$
$f'''(x_i) = \dfrac{-f(x_{i+3}) + 8f(x_{i+2}) - 13f(x_{i+1}) + 13f(x_{i-1}) - 8f(x_{i-2}) + f(x_{i-3})}{8h^3}$	$O(h^4)$
Fourth Derivative	
$f''''(x_i) = \dfrac{f(x_{i+2}) - 4f(x_{i+1}) + 6f(x_i) - 4f(x_{i-1}) + f(x_{i-2})}{h^4}$	$O(h^2)$
$f''''(x_i) = \dfrac{-f(x_{i+3}) + 12f(x_{i+2}) + 39f(x_{i+1}) + 56f(x_i) - 39f(x_{i-1}) + 12f(x_{i-2}) + f(x_{i-3})}{6h^4}$	$O(h^4)$

"Three ways" to improve the accuracy of derivative estimates

- Incorporate higher-order terms that will employ more data points
- Decrease step size (**h↓**)
- **Richardson extrapolation** – two less accurate derivative estimates can be used to calculate a third, more accurate approximation

Richardson Extrapolation

- Recall in Chapter 20 Numerical Integration, we had:

Eqn 20.4 $$I = I(h_2) + \frac{1}{(h_1/h_2)^2 - 1}\left[I(h_2) - I(h_1)\right]$$

$I(h_1)$ and $I(h_2)$ are two estimates of the same integral with **2 different step sizes.**

$\boxed{\text{for a special case } h_2 = h_1/2}$

$$\Rightarrow \quad I = \frac{4}{3} I(h_2) - \frac{1}{3} I(h_1)$$

O(h⁴) ← O(h²)   O(h²)

$$I = \frac{16}{15} I(h_2) - \frac{1}{15} I(h_1)$$

$O(h^6) \leftarrow O(h^4) \quad O(h^4)$

$$I = \frac{64}{63} I(h_2) - \frac{1}{63} I(h_1)$$

$O(h^8) \leftarrow O(h^6) \quad O(h^6)$

Richardson Extrapolation

- In a similar fashion:

for a special case $h_2=h_1/2$

$D(h_1)$ and $D(h_2)$ are two estimates of the same derivative with **2 different step sizes.**

$$\Rightarrow \qquad D = \frac{4}{3}D(h_2) - \frac{1}{3}D(h_1)$$

O(h⁴) ← O(h²)   ***O(h²)***

$$D = \frac{16}{15}D(h_2) - \frac{1}{15}D(h_1)$$

$O(h^6) \leftarrow O(h^4) \qquad O(h^4)$

$$D = \frac{64}{63}D(h_2) - \frac{1}{63}D(h_1)$$

$O(h^8) \leftarrow O(h^6) \qquad O(h^6)$

Run a ***Romberg algorithm*** (Lab_10 problem 2) until the result falls **below an acceptable error criterion**.

Problem 1: “Unequally” Spaced Data

- One way to handle **nonequispaced data** is to fit a **Lagrange interpolating polynomial** to a set of adjacent points that bracket the location value at which you want to **evaluate the derivative**.
- As an example, using a second-order **Lagrange** polynomial to fit three points and taking its derivative yields an analytical form (a linear equation):

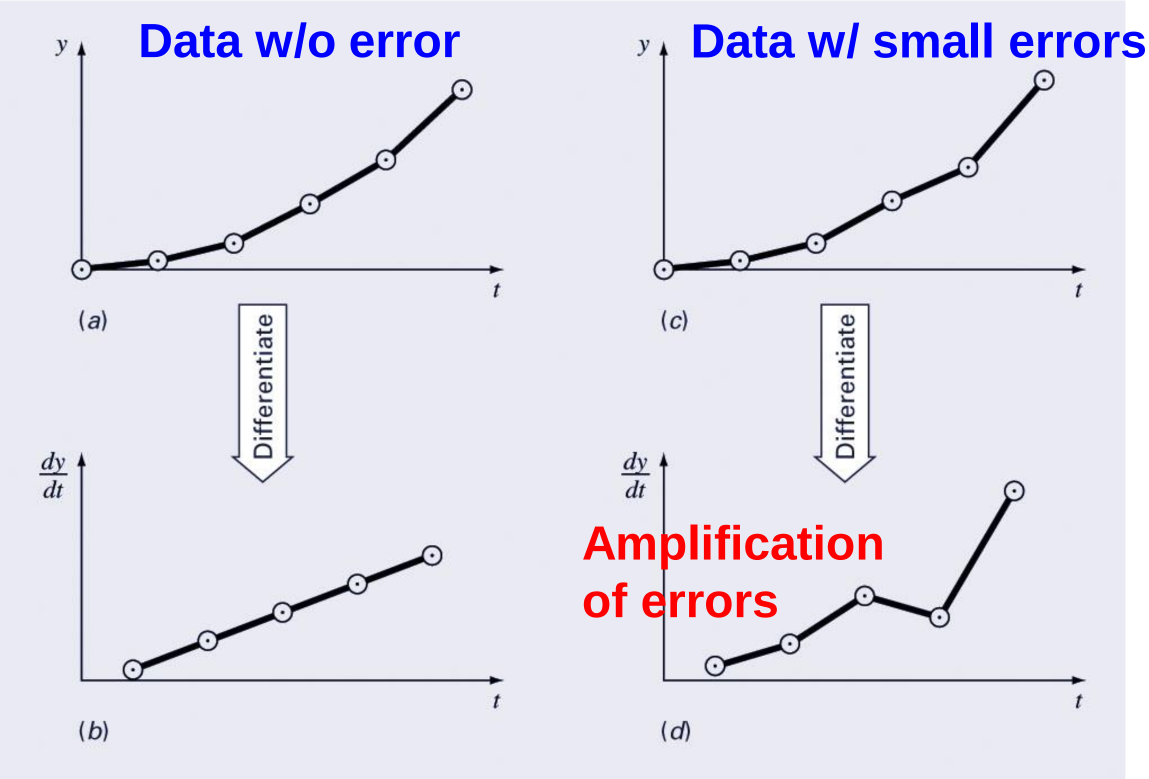
$$f'(x) = f(x_0)\frac{2x - x_1 - x_2}{(x_0 - x_1)(x_0 - x_2)} + f(x_1)\frac{2x - x_0 - x_2}{(x_1 - x_0)(x_1 - x_2)} + f(x_2)\frac{2x - x_0 - x_1}{(x_2 - x_0)(x_2 - x_1)}$$

Error is ***$O(h^2)$***, the same as centered-difference estimate.

Follow example 21.6 on p. 531

Problem 2: Derivatives and Integrals for Data with Errors

- A **shortcoming** of numerical differentiation is that it tends to **amplify errors in data**, whereas integration tends to **smooth or attenuate** data errors.

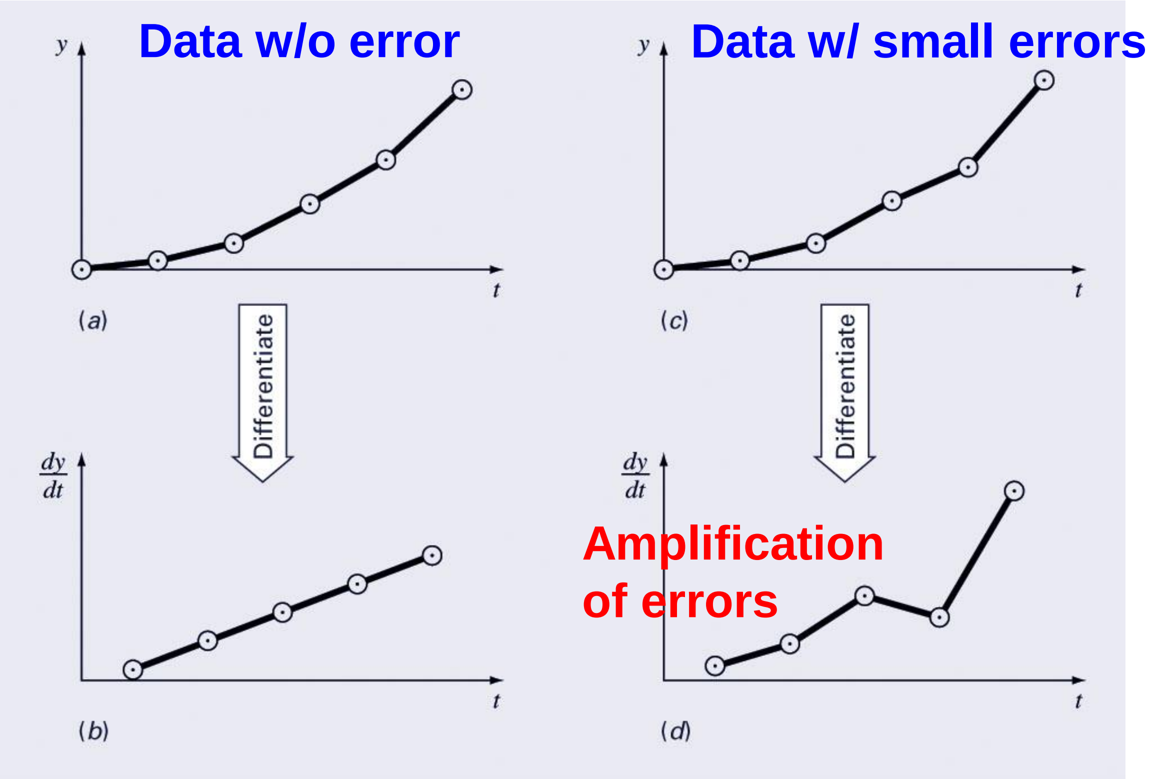


Why?

- Differentiation is **subtractive**.
  - random positive and negative errors tend to **add up**.
- Integration is a **summing process**.
  - random positive and negative errors **cancel out**.

Problem 2: Derivatives and Integrals for Data with Errors

- One approach for determining derivatives of imprecise data is to use least-squares regression to **fit a smooth, differentiable function** to the data!



Why?

- Differentiation is **subtractive**.
  - random positive and negative errors tend to **add up**
- Integration is a **summing process**.
  - random positive and negative errors **cancel out**

What is the best method to use?

Numerical Integration: trapezoidal, Simpson's 1/3, 3/8 rules, Richardson extrapolation, Gauss-Legendre, adaptive quadrature….

Numerical Differentiation: forward/background/centered finite divided differences**,** high-accuracy numerical differentiation formulas, Richardson extrapolation….

1. **Error:** what is truncation error that you want? (e.g. truncation error is of the order of h^2)

   $O(h)$ vs. $O(h^2)$, what price you pay?

2. **Accuracy:** what is accuracy that you want? (e.g. third-order accurate)

   $f''(\xi)$ (trapezoidal) vs. $f^{(4)}(\xi)$ (2-point G-L), what price you pay?

3. **Data format:** Equally spaced? On one side of x_i? Even or odd segments?

4. **Special circumstances:**

   Less accurate estimates are already available…
   A function has regions of abrupt changes, small steps must be used

Numerical Differentiation in MATLAB (p. 533)

- MATLAB has **two built-in functions** to help take derivatives, ***`diff`*** and ***`gradient`***:
- `diff(`*x*`)`
  - Returns the **difference between adjacent elements** in `x` (note: diff (x) is a vector of length n-1)
- `diff(`*y*`)./diff(`*x*`)`
  - Returns the difference between adjacent values in **`y`** divided by the **corresponding difference** in adjacent values of **`x`**

See a divided-difference approximation example on p. 534

Numerical Differentiation in MATLAB

- `fx = gradient(f, h)`
  Determines the **derivative of the data in `f`** at each of the points. The program uses forward difference for the first point, backward difference for the last point, and centered difference for the interior points. `h` is the spacing between points; if omitted `h`=1.
- The major advantage of `gradient` over `diff` is `gradient`'s result has the same size (length = n) as the original data.
- Gradient can also be used to find **partial derivatives** for matrices (see 21.8 case study on p. 539):
  `[fx, fy] = gradient(f, h)`

Partial Derivatives and Visualizing Fields (p. 538)

- **Mountain elevation** can be expressed by the following two-dimensional function:

$z = f(x, y)$, where z is the elevation

$$\frac{\partial f}{\partial x} = \lim_{\Delta x \to 0} \frac{f(x + \Delta x, y) - f(x, y)}{\Delta x}$$

$$\frac{\partial f}{\partial y} = \lim_{\Delta y \to 0} \frac{f(x, y + \Delta y) - f(x, y)}{\Delta y}$$

If you are mountain climbing and you stop at (x_0, y_0),

- the **slope** on your **east** side will be $\dfrac{\partial f(x_0, y_0)}{\partial x}$
- the **slope** on your **north** side will be $\dfrac{\partial f(x_0, y_0)}{\partial y}$

Partial Derivatives and Visualizing Fields (p. 538)

- **Mountain elevation** can be expressed by the following two-dimensional function:

  z = f (x, y), where z is the elevation

- For mountain climbing, you may like to know **the maximum slope** around you and the **direction** of that.

$$\nabla f = \frac{\partial f}{\partial x}\boldsymbol{i} + \frac{\partial f}{\partial y}\boldsymbol{j}$$

$$\nabla f = u\,\boldsymbol{i} + v\,\boldsymbol{j}$$

- ∇f is the "gradient of *f* "
- This vector represents the **steepest slope** around you
- The slope magnitude is

$$\sqrt{\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2}$$

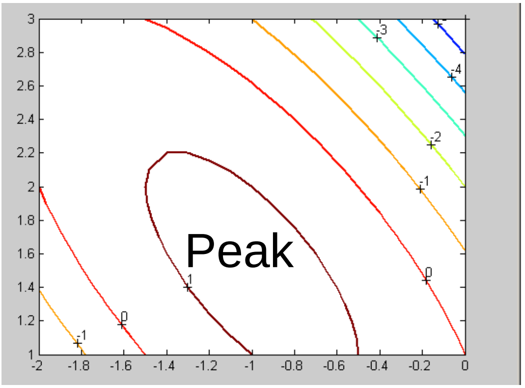
- Direction

$$\theta = \tan^{-1}\left(\frac{\partial f/\partial y}{\partial f/\partial x}\right)$$

Partial Derivatives and Visualizing Fields (p. 538)

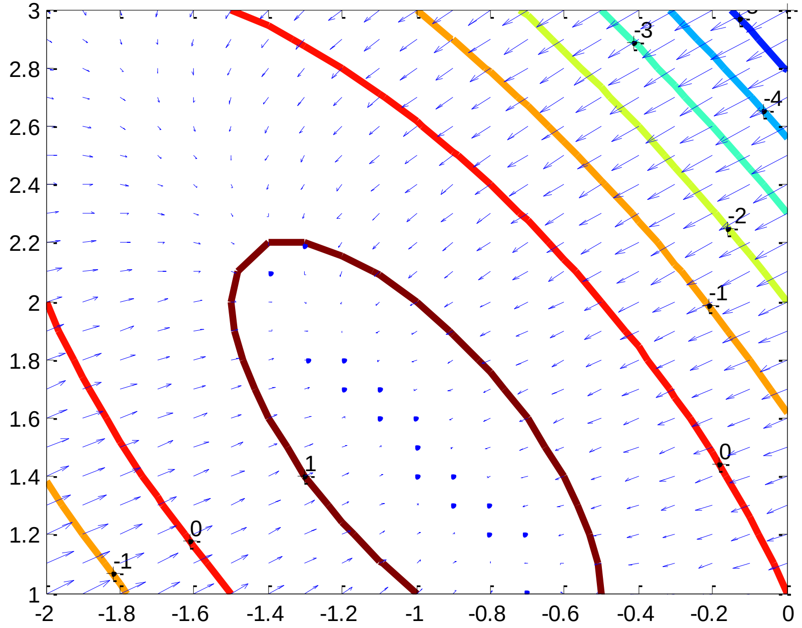
- MATLAB function `quiver (x, y, u, v)` allows us to draw "the steepest route to the peak" (i.e. the gradient field of *f*) at each point. $f(x, y) = y - x - 2x^2 - 2xy - y^2$ $\nabla f = u\,\boldsymbol{i} + v\,\boldsymbol{j}$

```
f=@(x,y) y-x-2*x.^2-2.*x.*y-y.^2;
[x,y]=meshgrid(-2:.1:0, 1:.1:3);
z=f(x,y);
[u,v]=gradient(z,0.1);

figure(1);
% contour(x,y,z) generates contour plot
cs=contour(x,y,z, 'linewidth', 2);clabel(cs);hold
on
% +u, +v indicate "the steepest rout to the peak"
quiver(x,y,u,v);hold off
```

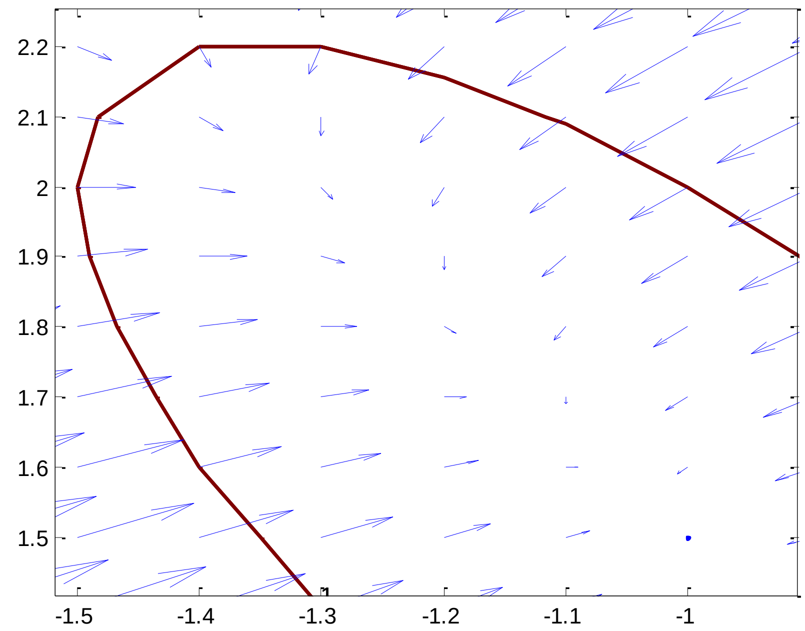


Partial Derivatives and Visualizing Fields (p. 538)

`quiver` generates “vector fields”
(here, the **gradient vector**)



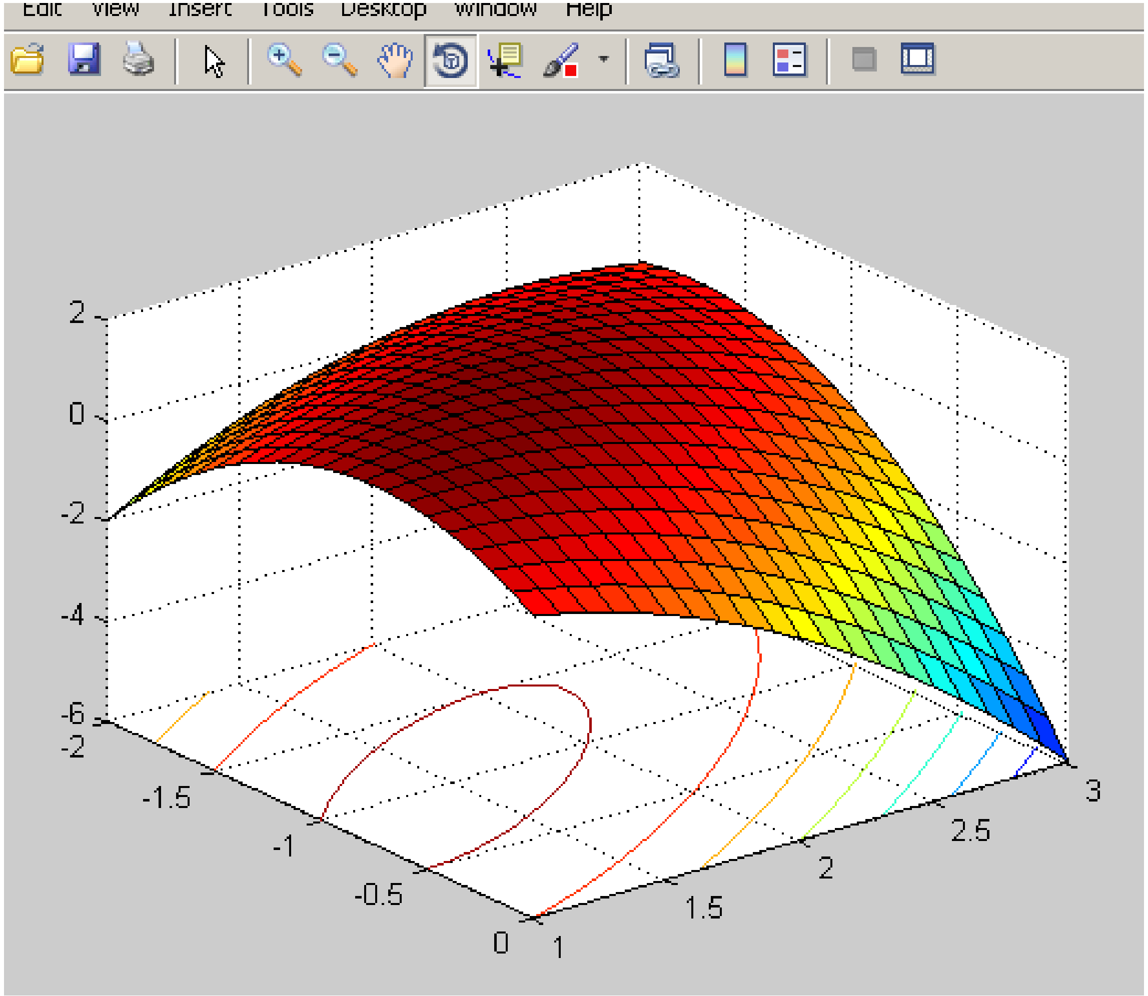
Zoom-in



If we draw the field in the opposite direction:

```
figure(2);
cs=contour(x,y,z, 'linewidth', 2);clabel(cs);hold on
quiver(x,y,-u,-v);hold off
% -u, -v indicate "how the ball travels as it rolls downhill"
```

surfc(x,y,z) generates a contour plot under the 3D surface plot



See *Quiver_example.m*

Slope Field (Direction Field, not in textbook)

- A **slope field** can provide information about the solution to a differential equation **without actually solving it**.

- Let consider the following first order differential equation

$$\frac{dy}{dx} = -y$$

The analytical solution is: $y = c \cdot e^{-x}$

- Let's make y' = f (x, y)

- Let's plot the vector **(1, f(x,y))** in (x,y) space. The resulting plot is called the **slope field**.

Try lecture011a.m

```
% Slope Field
   help dfield

 % Let consider simple function dy/dx=-y
   [dx, dy] = dfield('dfunS', [-3 3 -3 3], 31);
   axis([-3 3 -3 3]);
   axis('square')
```

dfield.m
plots the slope
field (1,f)

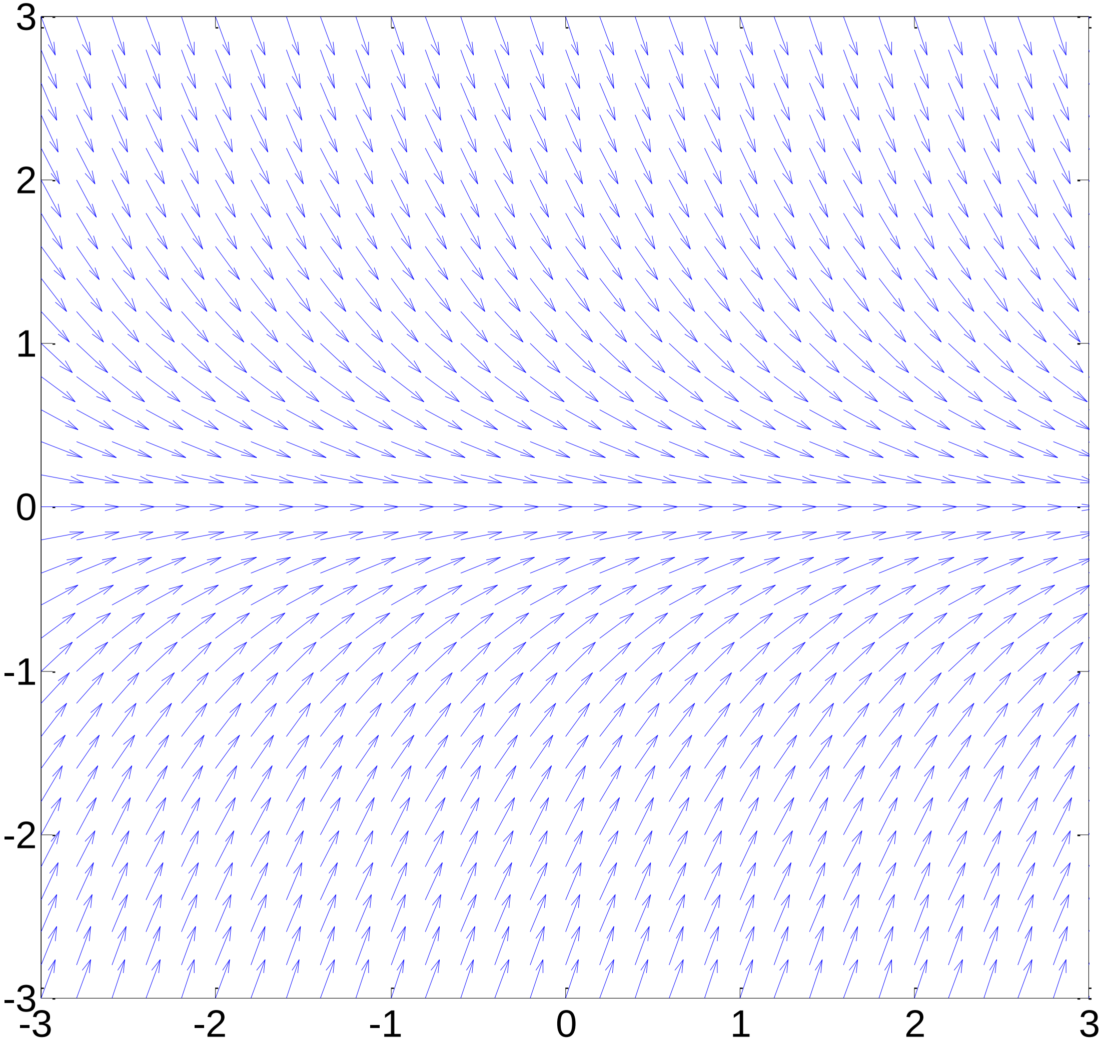
```
function f = dfunS(x,y)
% DFUNS(T,Y) is an example of the function
%   dy/dx = f(x,y)
% to be used with dfield (slope field)


f = -y;
```

MATLAB code: function dfield.m

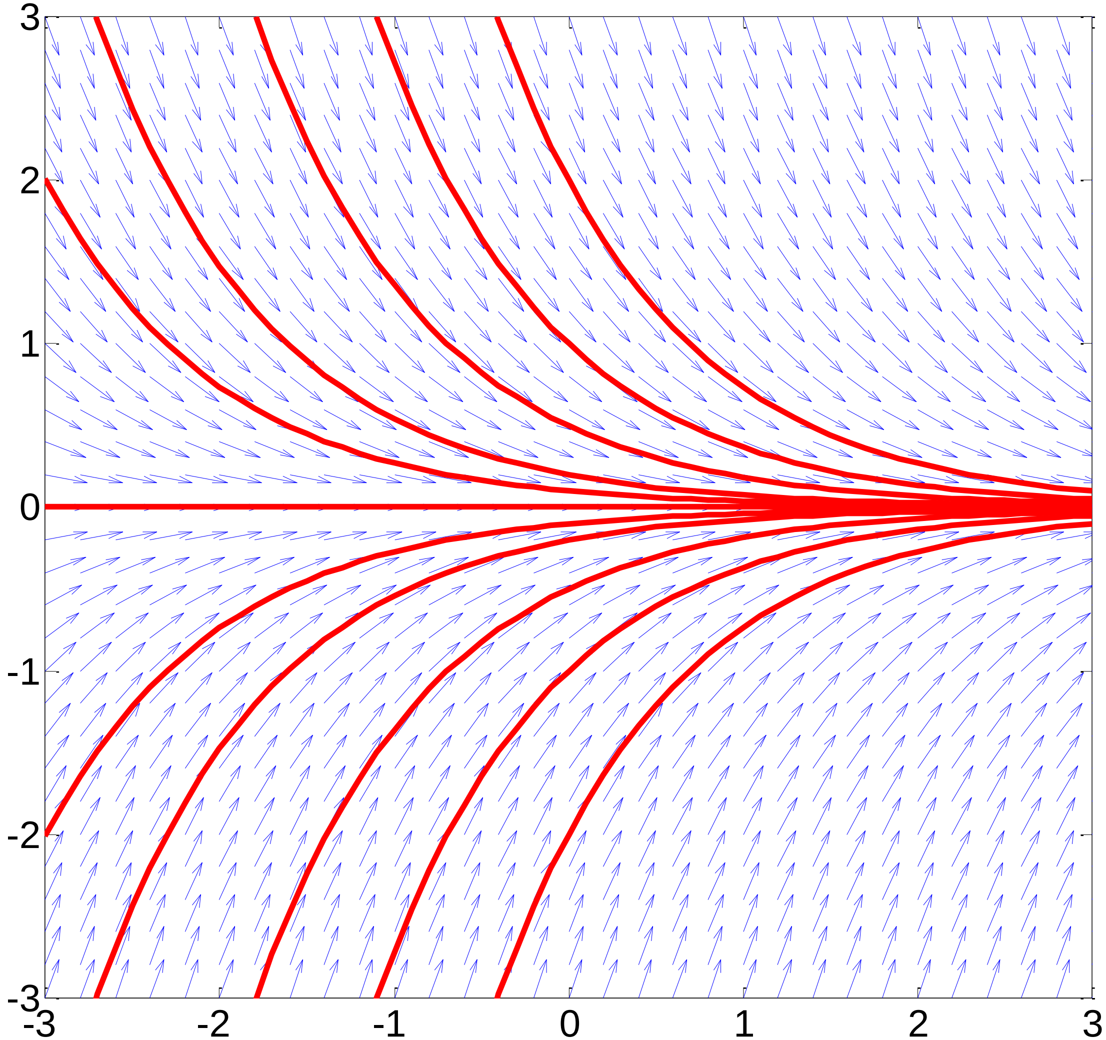
dfield.m
plots the slope field (1,f)

```
%   DFIELD.M   Plot slope field
%   dfield(fn,rm,Np)
%
%   The inputs to the function are:
%   fn = string name of the file containing the
%   function
%   rm = 4-element vector containing [minx maxx
%   miny maxy]
%   Np = number of points in x and y directions

      function [dx,dy] = dfield(fn,rm,Np)
%   compute x,y grid
      xx = linspace(rm(1),rm(2),Np);
      yy = linspace(rm(3),rm(4),Np);
      [x,y] = meshgrid(xx,yy);
%
%   evaluate function f(x,y) at each mesh point
      eval(['f = ',fn,'(x,y);'])
%
%   Tim Yeh: (dx,dy) is the "normalized vector"
%   of (1,f). Remember slope
%   field is a plot of (1,f) vector in xy space

      angle = atan(f);   dx = cos(angle);   dy =
sin(angle);
%
%   eliminate complex components
      dx(find(imag(dx)~=0)) = NaN;
      dy(find(imag(dy)~=0)) = NaN;
%
%   use QUIVER to plot the slope field for
%   f(x,y)
     quiver(x,y,dx,dy, 0.75);
%   end of function
```

Slope Field of y'=-y



Solutions using Slope Field ($y=c \cdot e^{-x}$)



Another Example

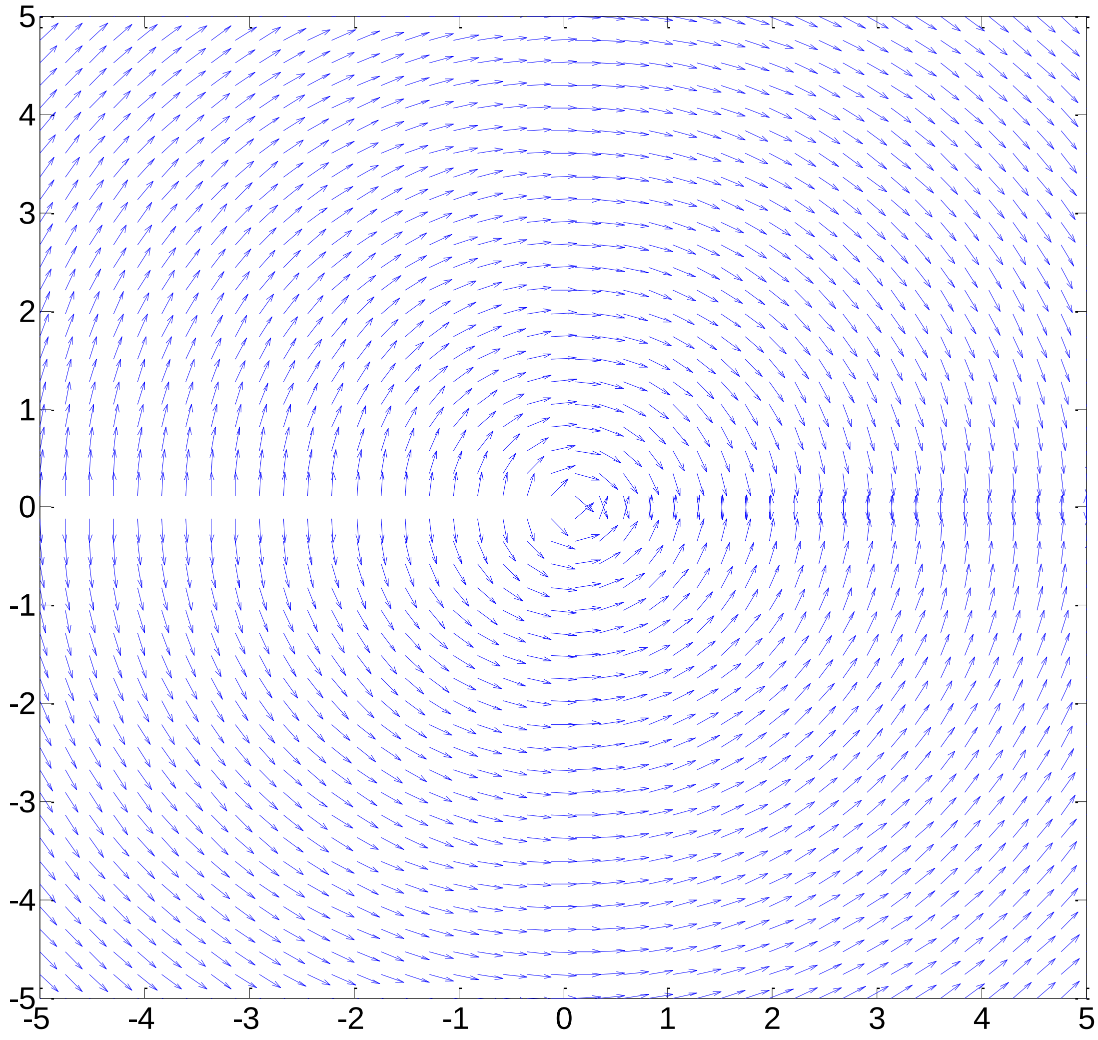
- Consider another first order differential equation

$$\frac{dy}{dx} = -\frac{x}{y}$$

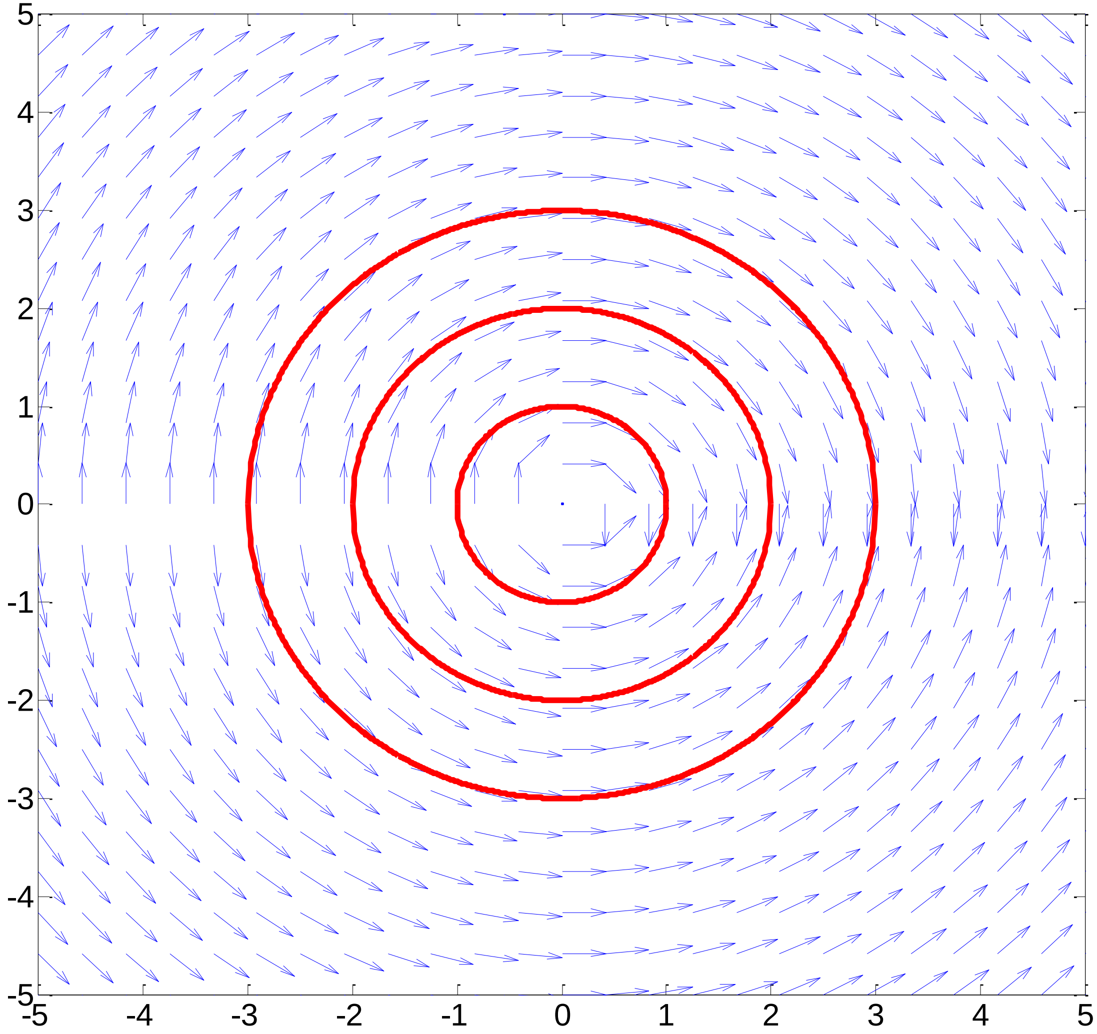
- Solution

  Circles of various radii (x^2+y^2 = C)

Slope Field of y'= -x / y



Solutions $x^2+y^2 = C$



Differential Equations

- First order differential equation

$$\frac{dy}{dx} = 2x + x^2 - 1 - e^x + y^2$$