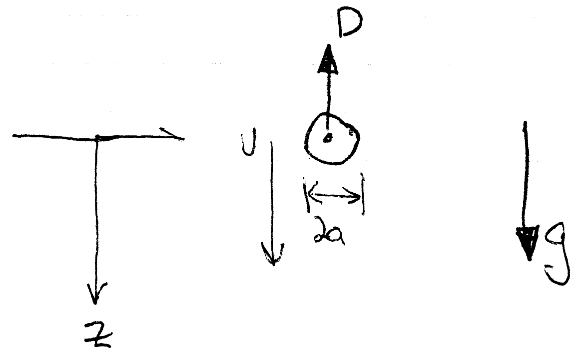
# Backward Euler example: Particle in free-fall

Consider a spherical particle falling through the atmosphere:

$$D = \frac{1}{2}\rho_g \pi a^2 U^2 C_D(Re)$$

$$Re = \frac{\rho_g U \, 2a}{\mu_g}$$




$$C_D = \frac{24}{Re} + \frac{6}{1+\sqrt{Re}} + 0.4$$

at $t=0$, $z=0$, $U=0$

## Governing equations:

$$m_p \dot{U} = m_p g - D$$

$$\dot{z} = U$$

So, in our canonical form:

$$\dot{U} = g - \frac{1}{m_p} D = f_U(U)$$

$$\dot{z} = U = f_z(U)$$

Or

$$\dot{\vec{U}} = \vec{F} \quad \text{where } \vec{U} = \begin{bmatrix} U \\ z \end{bmatrix}$$

$$\vec{F} = \begin{bmatrix} g - \frac{1}{m_p} D \\ U \end{bmatrix}$$

As described in the previous notes, we need $\frac{\partial \vec{F}}{\partial \vec{U}}$. ~~Here's that.~~ To do this, I would strongly recommend using the chain rule to keep this clean.

First, $f_v = g - \frac{1}{m_p} D$

Note, $f_v$ does not depend on $z$

$$\Rightarrow \boxed{\frac{\partial f_v}{\partial z} = 0}$$

The harder one is $\frac{\partial f_v}{\partial v}$:

$$\frac{\partial f_v}{\partial v} = \frac{\partial}{\partial v}\left(g - \frac{1}{m_p} D\right)$$

$$\boxed{\frac{\partial f_v}{\partial v} = -\frac{1}{m_p}\frac{\partial D}{\partial v}}$$

$$D = \frac{1}{2}\rho_g \pi a^2 v^2 C_D(Re)$$

$$\Rightarrow \frac{\partial D}{\partial v} = \frac{1}{2}\rho_g \pi a^2 v^2 \frac{\partial C_D}{\partial Re}\frac{\partial Re}{\partial v} + \frac{1}{2}\rho_g \pi a^2 (2v)\, C_D(Re)$$

$$\frac{\partial Re}{\partial v} = \frac{\partial}{\partial v}\left(\frac{\rho_g v\, 2a}{\mu_g}\right) = \frac{2a\rho_g}{\mu_g}$$

$$\frac{\partial C_D}{\partial Re} = \frac{\partial}{\partial Re}\left[\frac{24}{Re} + \frac{6}{1+\sqrt{Re}} + 0.4\right] = -\frac{24}{Re^2} - 6\left(1+\sqrt{Re}\right)^{-2}\frac{1}{2}Re^{-\frac{1}{2}}$$

Then $f_z = v \Rightarrow \frac{\partial f_z}{\partial z} = 0 \qquad \frac{\partial f_z}{\partial v} = 1$

$$\Rightarrow \frac{\partial \vec{f}}{\partial \vec{v}} = \begin{bmatrix} -\frac{1}{m_p}\frac{\partial D}{\partial v} & 0 \\ 1 & 0 \end{bmatrix}$$

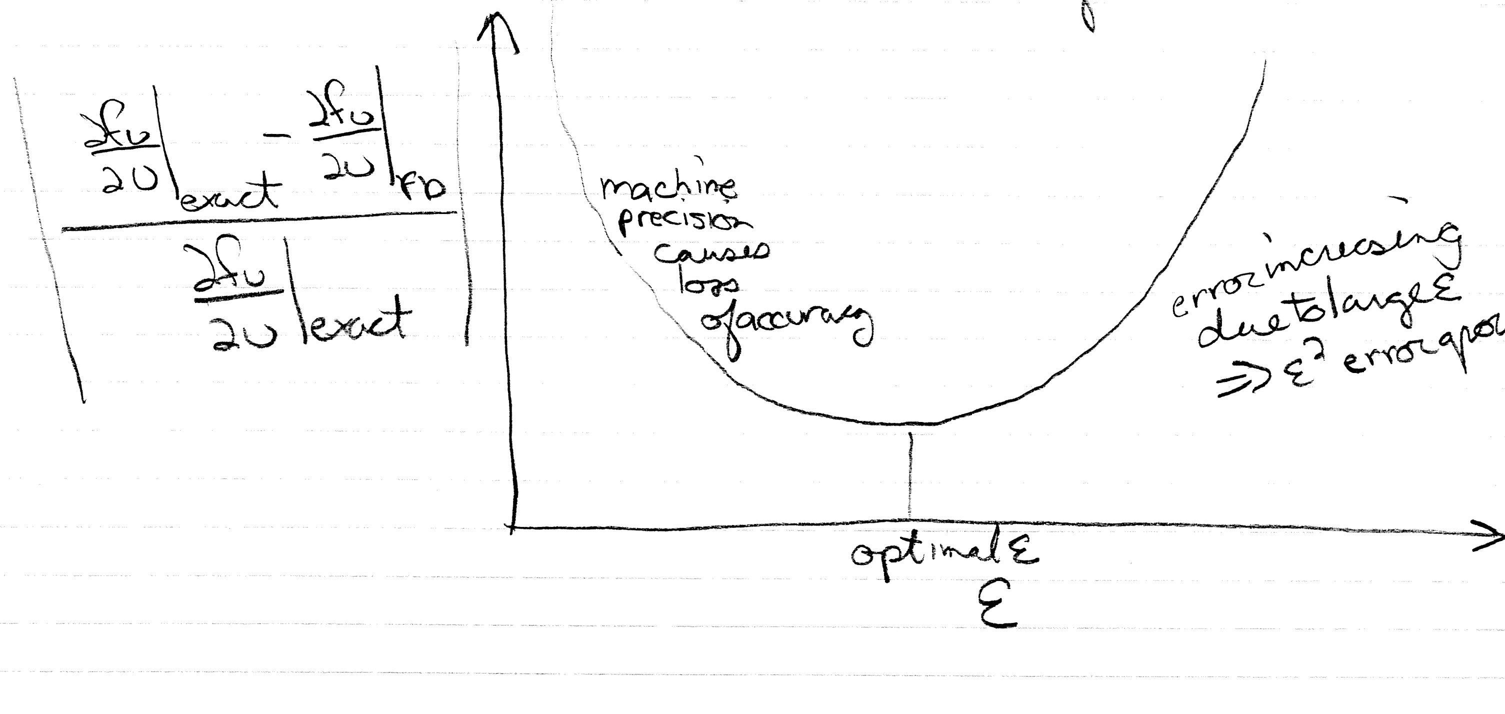
In some case, calculation of the derivatives $\frac{\partial f_U}{\partial U}$ may be extremely difficult to do by hand even with careful chain rule application.
In that case, finite differencing may be used to find these derivatives:

For example,

$$\left.\frac{\partial f_U}{\partial U}\right|_{FD} = \frac{f_U(U+\varepsilon) - f_U(U-\varepsilon)}{2\varepsilon} = \frac{\partial f_U}{\partial U} + \underbrace{O(\varepsilon^2)}_{\text{error}}$$

where $\varepsilon$ is a small number typically $\varepsilon \approx 1\times10^{-}$

Warning: There is some judgement in choosing $\varepsilon$.
* If taken too large, the derivative will not be accurate.
* If taken too small, the precision of the computer will be run into and the derivatives will again ~~lose~~ loose accuracy



Typically, the optimal $\varepsilon$ is around

$\varepsilon_{opt} \cong \sqrt{\text{Machine precision}}$.

Comments on using finite differenced $\frac{\partial \vec{f}}{\partial \vec{U}}$:

* Very useful for debugging! If $\vec{f}(\vec{U},t)$ is difficult to differentiate, than comparing $\left.\frac{\partial \vec{f}}{\partial \vec{U}}\right|_{FD}$ and a hand-coded $\frac{\partial \vec{f}}{\partial \vec{U}}$ is an essential tool.

* Finite differences are usually (though not always) slower than hand-coded derivatives. Though, with careful implementation of finite differences, the cost is usually only 3-4 times hand-coding.

* If you are using a blackbox integrator (such as in matlab), the differentiation will be done w/ finite differences, but, usually the user can supply the derivatives if desired.

```
clear all;

% Initialize some parameters
rhog = 0.9; % Density of air at 3000m
a    = 0.01; % Particle radius
mu   = 1.693e-5; % Viscosity of air at 3000m
g    = 9.8; % Gravity
mp   = 917*4*pi/3*a^3; % mass of ice particle

% Initial conditions
u0 = 0.001; % Use a small initial velocity to avoid problems with Re=0 and CD
z0 = 0; % Initial particle location

% Set time length of integration, and number of steps
Tmax = 25;
N = 100;
dt = Tmax/N;

f = zeros(2,1);
v(:,1) = [u0; z0];
% Start iterative loop
for n = 2:N+1,

% Calculate drag at n-1
  u = v(1,n-1);
  Re = rhog*u*(2*a)/mu;
  CD = 24/Re + 6/(1+sqrt(Re)) + 0.4;
  D = 0.5*rhog*u^2*pi*a^2*CD;

% Calculate right-hand sides at n-1
  f(1) = g - D/mp;
  f(2) = u;

% Update using Forward Euler
  v(:,n) = v(:,n-1) + dt*f;

end

% Plot results
t = linspace(0,Tmax,N+1);
u = v(1,:);
z = v(2,:);

subplot(211);
plot(t,z);
xlabel('time');ylabel('z');
title('Forward Euler integration');

subplot(212);
plot(t,u);
xlabel('time');ylabel('u');
```

```
clear all;

% Initialize some parameters
rhog = 0.9; % Density of air at 3000m
a    = 0.01; % Particle radius
mu   = 1.693e-5; % Viscosity of air at 3000m
g    = 9.8; % Gravity
mp   = 917*4*pi/3*a^3; % mass of ice particle

% Initial conditions
u0 = 0.001; % Use a small initial velocity to avoid problems with Re=0 and CD
z0 = 0; % Initial particle location

% Set time length of integration, and number of steps
Tmax = 25;
N = 100;
dt = Tmax/N;

% Set Newton-Raphson convergence parameters
Mmax = 10;
restol = 1e-4;

% Initialize vector for ODE integration
f = zeros(2,1);
w = zeros(2,1);
res = zeros(2,1);
v(:,1) = [u0; z0];

% Start iterative loop
for n = 2:N+1,

% Begin sub-iteration loop
  m = 0;
  curres = restol + 1; % Doing this forces at least one sub-iteration to occur
  w = v(:,n-1);
  while ((m < Mmax) & (curres > restol)),

    u = w(1);
    Re     = rhog*u*(2*a)/mu;
    pRe_pu = rhog*  (2*a)/mu;

    CD      = 24/Re + 6/(1+sqrt(Re)) + 0.4;
    pCD_pRe = -24*Re^(-2) - 6*(1+sqrt(Re))^(-2)*0.5/sqrt(Re);

    D     = 0.5*rhog*u^2*pi*a^2*CD;
    pD_pu = 0.5*rhog*u^2*pi*a^2*pCD_pRe*pRe_pu + rhog*pi*a^2*u*CD;

% Calculate right-hand sides using w
    f(1) = g - D/mp;
    f(2) = u;

% Calculate residual
    res = w - v(:,n-1) - dt*f;
    curres = norm(res);

% Calculate linearization of f with respect to w
    pf_pw(1,1) = -pD_pu/mp; % this is pfu/pu
    pf_pw(1,2) = 0.0;       % this is pfu/pz
    pf_pw(2,1) = 1.0;       % this is pfz/pu
    pf_pw(2,2) = 0.0;       % this is pfz/pz

% Calculate linearization of residual with respect to w
    pres_pw = eye(size(pf_pw)) - dt*pf_pw;

% Calculate dw from pres_dw*dw = -res
    dw = -pres_pw\res;
    w  = w + dw;

% Increment sub-iteration counter
    m = m + 1;

  end

  % Check on convergence and update states
  if (curres > restol),
    fprintf('Maximum number of Newton sub-iterations occurred\n');
  end

  v(:,n) = w;

end

% Plot results
t = linspace(0,Tmax,N+1);
u = v(1,:);
z = v(2,:);

subplot(211);
plot(t,z);
xlabel('time');ylabel('z');
title('Backward Euler integration');

subplot(212);
plot(t,u);
xlabel('time');ylabel('u');
```

```
clear all;

% Initialize some parameters
rhog = 0.9; % Density of air at 3000m
a    = 0.01; % Particle radius
mu   = 1.693e-5; % Viscosity of air at 3000m
g    = 9.8; % Gravity
mp   = 917*4*pi/3*a^3; % mass of ice particle

% Store the parameters in a vector for later use
p = [rhog, a, mu, g, mp];

% Initial conditions
u0 = 0.001; % Use a small initial velocity to avoid problems with Re=0 and CD
z0 = 0; % Initial particle location

% Set time length of integration, and number of steps
Tmax  = 25;
N     = 100;
dt    = Tmax/N;
eps   = 1e-4; % Finite difference step size for pf_pw calculation

% Set Newton-Raphson convergence parameters
Mmax = 10;
restol = 1e-4;

% Initialize vector for ODE integration
f = zeros(2,1);
w = zeros(2,1);
res = zeros(2,1);
v(:,1) = [u0; z0];

% Start iterative loop
for n = 2:N+1,

% Begin sub-iteration loop
  m = 0;
  curres = restol + 1; % Doing this forces at least one sub-iteration to occur
  w = v(:,n-1);
  while ((m < Mmax) & (curres > restol)),

% Calculate rhs
    f = drop_rhs(w, p);

% Calculate residual
    res = w - v(:,n-1) - dt*f;
    curres = norm(res);

% Calculate linearization of f with respect to w using finite differences
    for ii = 1:2,
      w0 = w(ii);

      w(ii) = w0 + eps; % Perturb ii state by eps
      fp = drop_rhs(w ,p);

      w(ii) = w0 - eps; % Perturb ii state by -eps
      fm = drop_rhs(w ,p);

      w(ii) = w0; % Restore ii state to original value

      pf_pw(:,ii) = (fp - fm)/(2*eps); % Finite difference

    end

% Calculate linearization of residual with respect to w
    pres_pw = eye(size(pf_pw)) - dt*pf_pw;

% Calculate dw from pres_dw*dw = -res
    dw = -pres_pw\res;
    w  = w + dw;

% Increment sub-iteration counter
    m = m + 1;

  end

  % Check on convergence and update states
  if (curres > restol),
    fprintf('Maximum number of Newton sub-iterations occurred\n');
  end

  v(:,n) = w;

end

% Plot results
t = linspace(0,Tmax,N+1);
u = v(1,:);
z = v(2,:);

subplot(211);
plot(t,z);
xlabel('time');ylabel('z');
title('Backward Euler integration with finite-differenced pf/pw');

subplot(212);
plot(t,u);
xlabel('time');ylabel('u');
```

```
function [f] = drop_rhs(w, p)

% Returns rhs of particle equation given the current state (w)
% and the parameters of the problem (p)

% Unpack the parameters from p
rhog = p(1);
a    = p(2);
mu   = p(3);
g    = p(4);
mp   = p(5);

% Get velocity
u = w(1);

% Calculate drag
Re = rhog*u*(2*a)/mu;
CD = 24/Re + 6/(1+sqrt(Re)) + 0.4;
D  = 0.5*rhog*u^2*pi*a^2*CD;

% Calculate right-hand sides
f = zeros(2,1);
f(1) = g - D/mp;
f(2) = u;
```