Matlab for Oceanography
Meeting 3

Root finding
- chemical equilibrium
- rising/falling Argo float

Finding roots - Example problem 1

Finding roots - Example problem 1

Consider the reversible chemical reaction $2A+B \leftrightarrow C$ with equilibrium $K = [C]/([A]^2[B])$.

Given K and initial concentrations of all three species you want to know the equilibrium concentrations of these species.

Finding roots - Example problem 1

Consider the reversible chemical reaction $2A+B \leftrightarrow C$ with equilibrium $K = [C]/([A]^2[B])$.

Given K and initial concentrations of all three species you want to know the equilibrium concentrations of these species.

Assuming x is the number of mols of C that are produced, $K = (C_{ini}+x)/(A_{ini}-2x)^2/(B_{ini}-x)$.

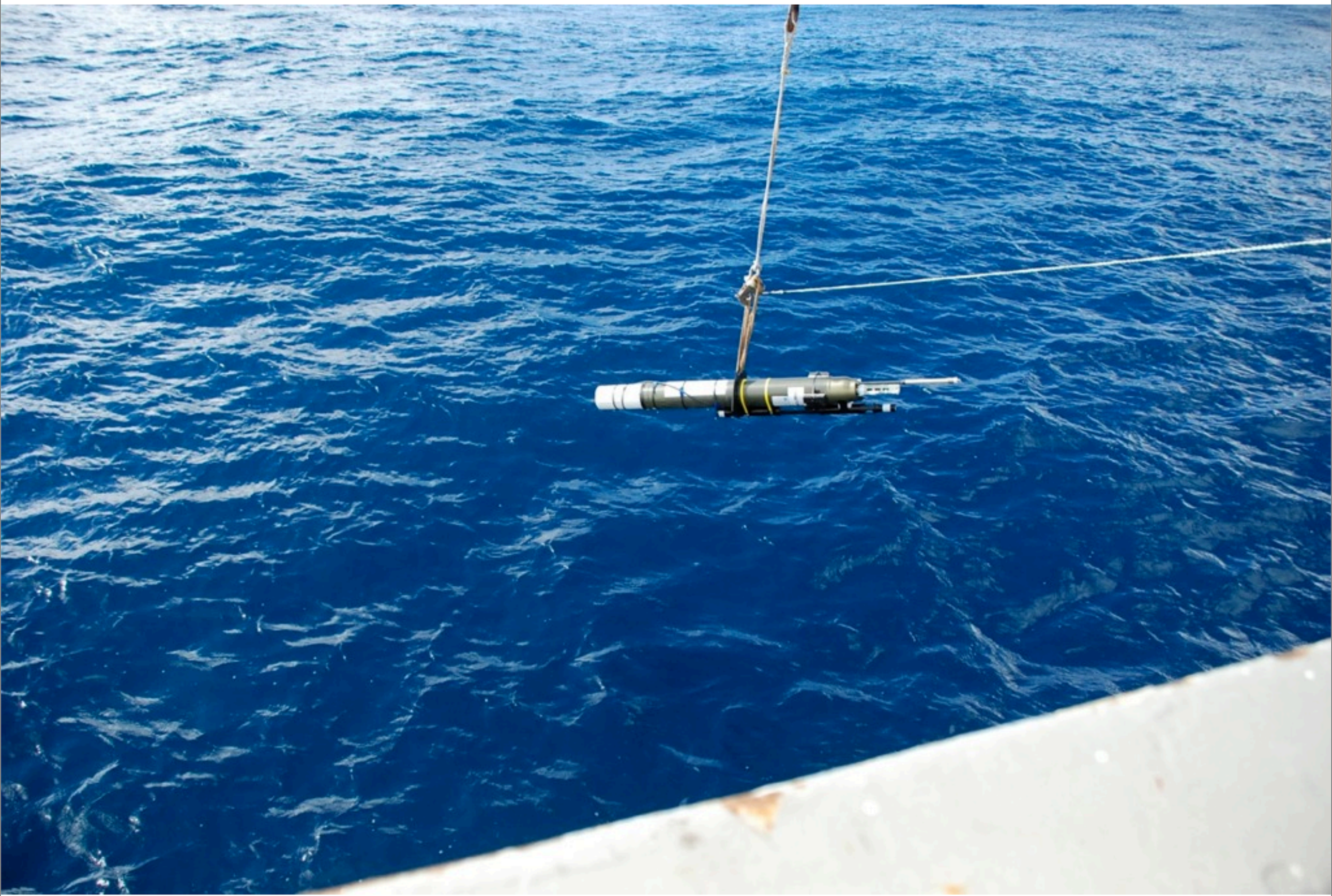
This is a third-order polynomial in x (can't be solved analytically).

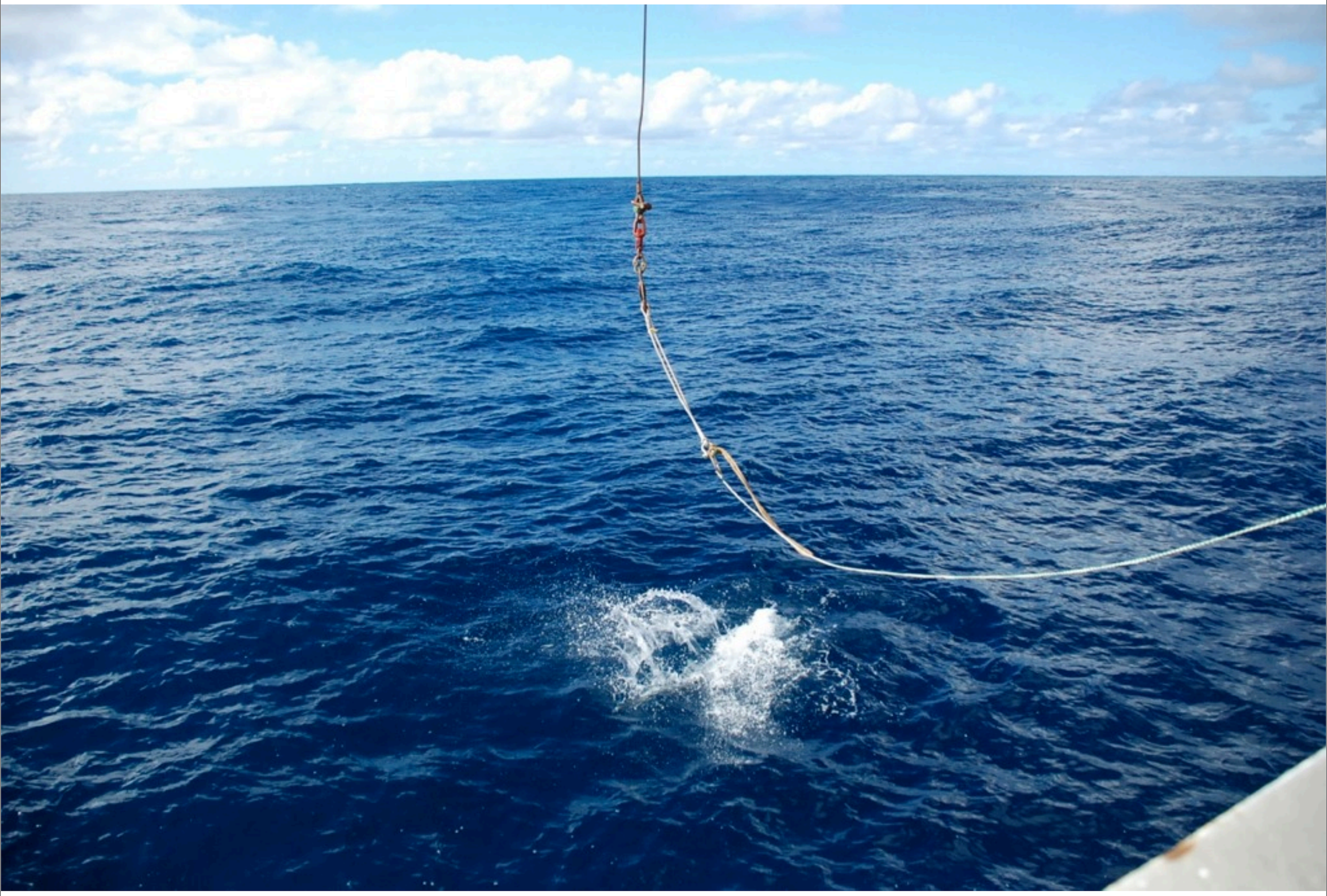
Finding roots - Example problem 2

Suppose you are determining the ballasting of a density-driven device (e.g. an Argo float) to achieve a certain fall rate (w) because of the sampling rate and response time of your sensors.

A common approximation of the falling/rising behavior such a device is

$$\frac{d^2z}{dt^2} = \frac{g(m - m_w)}{m} - \frac{\rho_w C_D A}{2m} w^2$$





Wednesday, October 12, 2011

gravitational acceleration

vertical position

water density

drag coefficient

effective surface area

$$\frac{d^2z}{dt^2} = \frac{g(m - m_w)}{m} - \frac{\rho_w C_D A}{2m} w^2$$

vertical velocity

time

mass of the device

mass of displaced water

This is a first-order ODE in *w* (fall velocity).

With initial conditions $w(t=0)=0$ the ODE can be solved analytically:

$$w(t) = \sqrt{\frac{g(m - m_w)}{C}} \tanh(\sqrt{\frac{gC(m - m_w)}{m^2}} t)$$

where $C = \frac{\rho_w C_D A}{2}$

It is not possible to isolate *m* for a given value of *w*.

With initial conditions $w(t=0)=0$ the ODE can be solved analytically:

$$w(t) = \sqrt{\frac{g(m - m_w)}{C}} \tanh(\sqrt{\frac{gC(m - m_w)}{m^2}} t)$$

where $C = \frac{\rho_w C_D A}{2}$

It is not possible to isolate m for a given value of w.

An alternative way of looking at this problem is this:

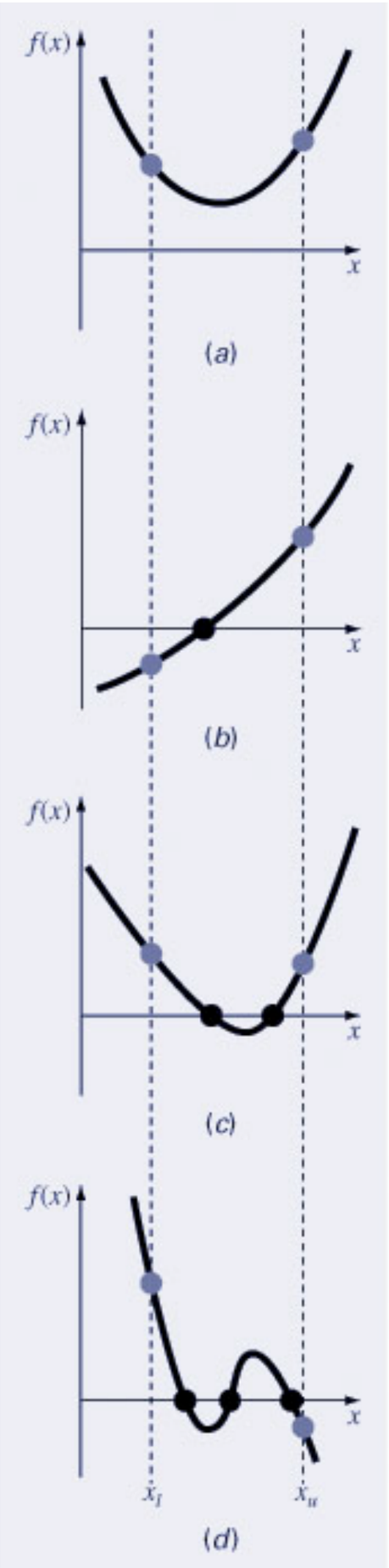
$$f(m) = \sqrt{\frac{g(m - m_w)}{C}} \tanh(\sqrt{\frac{gC(m - m_w)}{m^2}} t) - w(t)$$

We are looking for the value of m that makes $f(m) = 0$. We have transferred to problem into a root finding problem.

A simple method for obtaining the estimate of the root of the equation $f(x)=0$ is to make a plot of the function and observe where it crosses the x-axis.

Graphing the function can also indicate where roots may be and where some root-finding methods may fail. Some cases are:

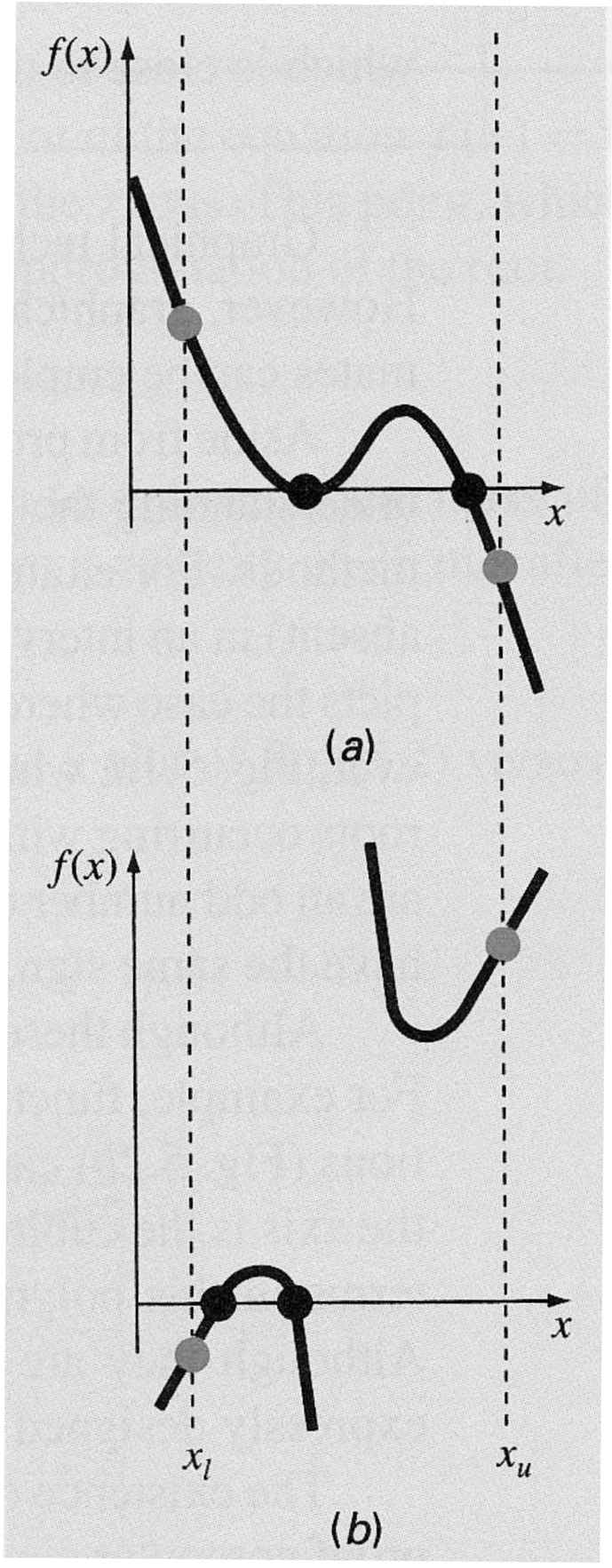
- Same sign, no roots
- Different sign, one root
- Same sign, two roots
- Different sign, three roots



Diversity of cases makes it difficult to develop foolproof numerical methods for root finding.

Special cases:

- Multiple roots
- Different sign, even number of roots



Bracketing Methods

Bracketing methods are based on making two initial guesses that "bracket" the root - that is, are on either side of the root.

Brackets are formed by finding two guesses x_l and x_u where the sign of the function changes; that is, where $f(x_l)\ f(x_u) < 0$

The *incremental search* method tests the value of the function at evenly spaced intervals and finds brackets by identifying function sign changes between neighboring points.

Let's look at an implementation of the incremental search method: "incsearch.m"

and find the roots of the following function
first graphically,
than by incremental search

$$f(x) = \sin(10x) - \cos(3x)$$

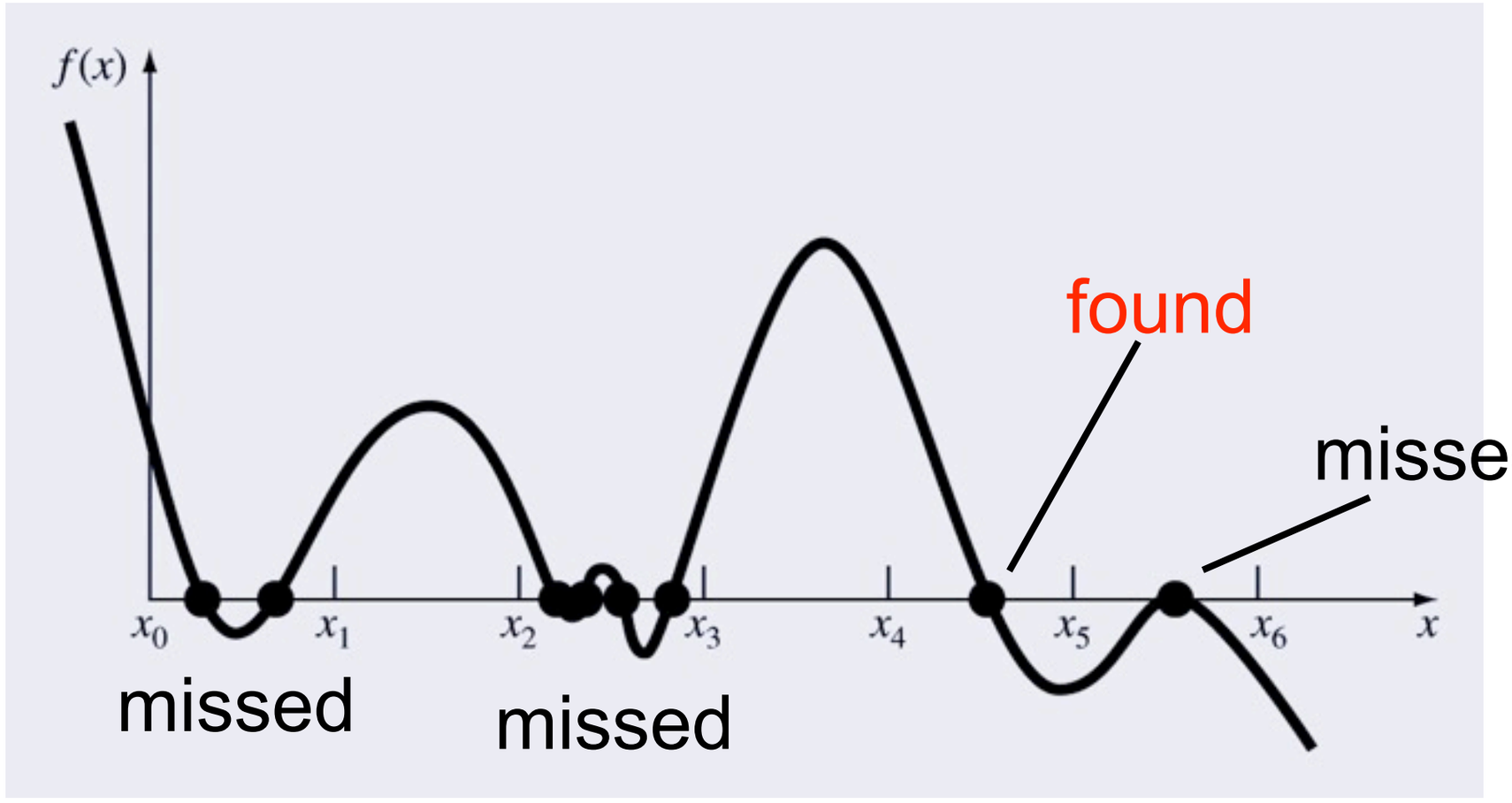
within the interval [3,6].

Look at my "Finding_roots_example1.m"

Pitfalls of Incremental Search

If the spacing between the points of an incremental search are too far apart, brackets may be missed due to capturing an even number of roots within two points.

Incremental searches cannot find brackets containing even-multiplicity roots.

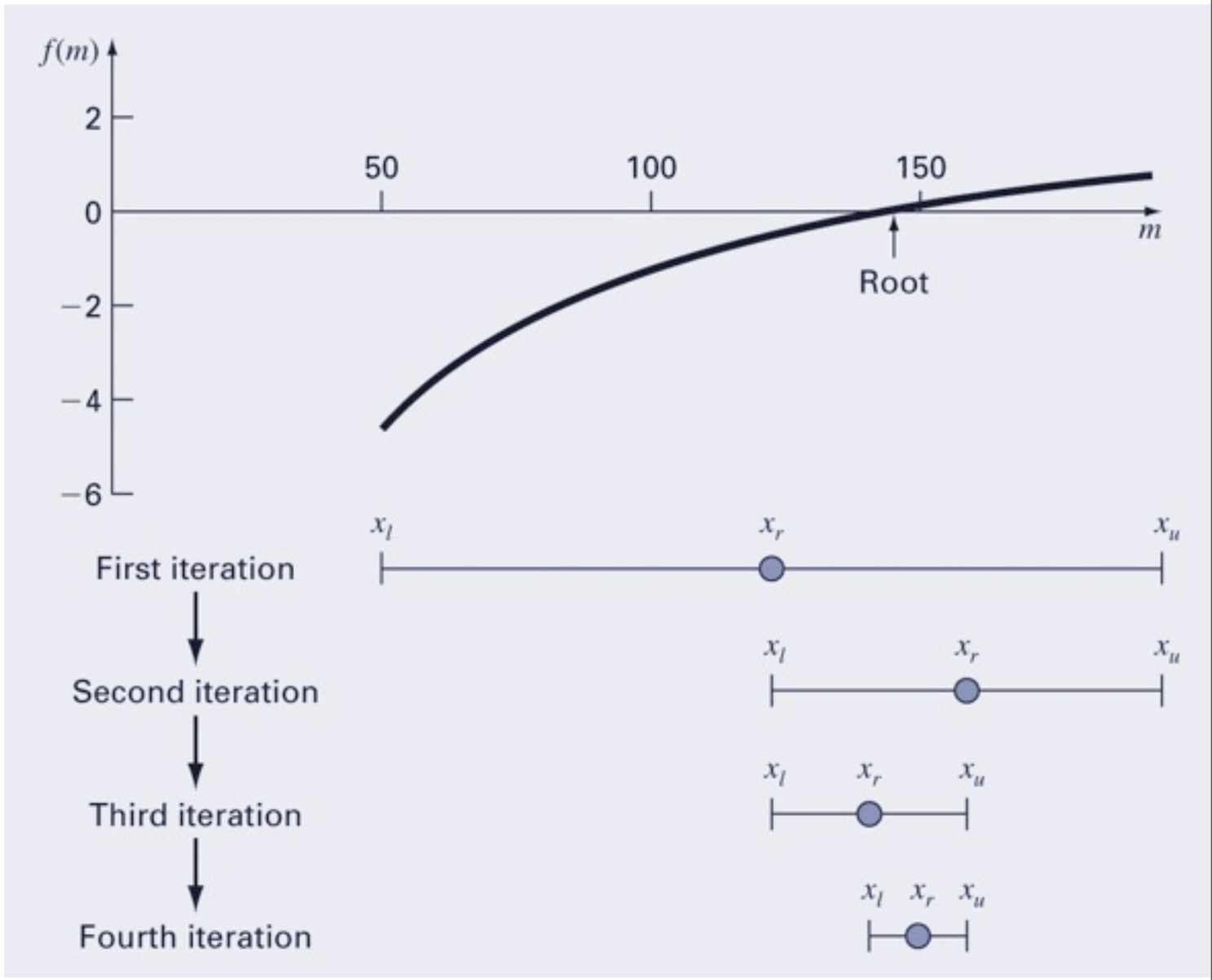


Bisection method

Is a variation of the incremental search method in which the interval is always divided in half.

If a function changes sign over an interval, the function value at the midpoint is evaluated.

The location of the root is then determined as lying within the subinterval where the sign change occurs.



The absolute error is reduced by a factor of 2 for each iteration.

We can use this to determine an upper limit on the absolute error of the approximation for every iteration.

The absolute error estimate is solely dependent on the absolute error at the start of the process (the space between the two guesses) and the number of iterations:

$$E_a^n = \frac{\Delta x^0}{2^n}$$

Thus the required number of iterations to obtain a particular absolute error can be calculated based on the initial guesses:

$$n = \log_2\left(\frac{\Delta x^0}{E_{a,d}}\right)$$

Let's look at an implementation of the bisection method: "bisect.m"

and then use it to determine the mass that a sinking Argo float would need to have to sink a speed of 2 m/s (at t = 10 s).

Use the following parameters:

density of seawater 1035 kg m^{-3}

drag coefficient $C_D = 0.25$

surface area $A = 0.2\ m^2$

mass of seawater $m_w = 207$ kg (assuming a float volume of 0.2 m^3)

Look at "Bisection_method_example.m"

Of course Matlab has built-in functions for root finding:

- **fzero** (general function that identifies the root closest to an initial guess or within and interval)
- **roots** (for polynomials)

fzero usage:

x = fzero(function,x0)

where x0 is an initial guess (scalar) or a 2-element vector that specifies a search interval (sign change has to occur)

x = fzero(function,x0,options)

where options is a structure created with optimset; allows to control level of output, termination criterion, etc.

Note that there is no obvious way of passing arguments to function (can be done with anonymous function handle though).

optimset usage:

options = optimset('para1',val1,'para2',val2,...)

will create options structure with values as specified (unspecified values will lead to default value being used)

Parameters for the option structure include:

Display - Level of display [off | iter | notify | final]
TolX - Termination tolerance on X [positive scalar]
FunValCheck - Check for invalid values, such as NaN or complex, from user-supplied functions [{off} | on]

Using anonymous function to pass arguments:

Parameter-dependent function:

```
function f = myfun(x,c)
f = cos(c*x);
```

First, set parameter, than use anonymous function with only one input argument:

```
c = 2;
x = fzero(@(x) myfun(x,c),0.1)
```

Now use fzero to solve the falling float problem.

See my script "Fzero_fallingfloat.m"

roots usage:

roots(C)
finds the roots of the polynomial
$f(x) = C(1)x^N + ... + C(N+1)$
assuming that C has N+1 elements.

Example: polynomial $f(x) = x^3-3x^2-x+3$

```
>> roots([1 -3 -1 3])
```

Let's return to example 1 from the beginning today:

Consider the reversible chemical reaction $2A+B \leftrightarrow C$

with equilibrium $K = [C]/([A]^2[B])$.

Assuming x is the number of mols of C that are produced,

$K = (C_{ini}+x)/(A_{ini}-2x)^2/(B_{ini}-x)$.

Solve this for $K=0.016$, $A_{ini} = 42$, $B_{ini} = 28$, and $C_{ini} = 4$ first graphically, then using one of the numerical methods we described. Make sure to check your solution.

See my script "Solving_example1.m"