10.34, Numerical Methods Applied to Chemical Engineering
Professor William H. Green


1D-Problem

\[ f(x) = 0 \quad Q_{\text{rxn}} \exp(-\frac{E_a}{RT}) + h(T - T_a) + c(T^4 - T_a^4) = 0 \]

unknown: \( T \) of reactor

\[ (+) \quad \text{heat of reaction} \]

\[ (-) \quad \text{convection} \]

\[ (-) \quad \text{radiation} \]

Gain heat

Lose heat

2 steady state temperatures

Make a plot with MATLAB

\[ f(T) \]

\[ T \]

Figure 1. 1D problem

*netheat.m*

function qdot = netheat(T)

% computes the net heating rate of a reactor
% qdot = 0 at the steady state

qdot = Q.*exp(-Ea/(R.*T)) + h.*(T-Ta) + c.*(T.^4-Ta.^4);

Q = -2e-5;
Ea = 5000;
R = 1.987;
h = 3;
Ta = 300;
c = 1e-8;

Tvec = linspace(300,3000)
qdot = netheat(Tvec)
plot(Tvec,qdot)

Figure 2. Professor Green modified variables \( Q \) and \( c \) until the plot looked like the one above. Increased \( Q \) and decreased \( c \).

To solve for steady state zeros

\[ f(T) = 0 \]

Figure 3. Have computer bracket in and find small range where plot goes from negative to positive.

Cite as: William Green, Jr., course materials for 10.34 Numerical Methods Applied to Chemical Engineering, Fall 2006. MIT OpenCourseWare (http://ocw.mit.edu), Massachusetts Institute of Technology. Downloaded on [DD Month YYYY].
**Bisection**

Start \(a, b\) such that \(f(a) < 0\) and \(f(b) < 0\). Let \(x = \frac{a + b}{2}\). If \(f(x) \cdot f(a) > 0\), then \(a = x\); otherwise, \(b = x\).

This is a problem of TOLERANCE if \((b-a) < \text{tol}\) stop.

**Absolute tolerance**
- atol: has units
- if \(|f(x)| < \text{atol} \cdot f\) has to be BIG number

**Relative tolerance**
- rtol: if \((b-a) < \text{rtol} \cdot |a|\)

---

**In MATLAB**

```matlab
function x = bisect(f,a,b,atolx,rtolx,atolf)
% solves \(f(x) = 0\)
while abs(b-a) > atolx
    x = 0.5*(b+a);
    if((feval(f,x)*feval(f,a)>0)
        a=x;
    else
        b=x;
    end
end
```

**Command Window**

```matlab
x = bisect(@netheat,300,2000,0.1,0,0)  
x = 1.2373e+003
```

**CHECK:** netheat(1237) = -1.0474

Keep in mind: never get actual solution, but can come close.

We can change tolerances to improve results.

```matlab
x = bisect(@netheat,300,2000,0.1,1e-2,0.5) 
x = 1.2363e+003
```

looser tolerance gives less accurate answer

---

Figure 4. Function must be continuous.

- Bisection cuts interval by 2 each time.
Every time we cut 3 times, we lose a sig fig

In bisection, time grows linearly with the number of significant figures.

\[ a < x^{\text{true}} < b \]
\[ x^{\text{true}} = x^{\text{soln}} \pm b-a/2 \]

**Newton’s Method (1-D)**

![Graph showing Newton's Method](image)

evaluates slope of \( f(x) \)

next guess is the \( x_{\text{new}} \) that satisfies \( f(x_{\text{new}})=0 \)

for a line from \( f(x_{\text{guess}}) \) with the slope at \( f(x_{\text{guess}}) \)

**Figure 5. Newton's Method.**

\[
f(x) = f(x_0) + f'(x_0)(x-x_0) + O(\Delta x^2)
\]

\[
0 = f(x_{\text{guess}}) + f'(x_{\text{guess}})(x-x_{\text{guess}})
\]

\[
x^{\text{new}} = x_{\text{guess}} - f(x_{\text{guess}})/f'(x_{\text{guess}})
\]

For a good guess Newton’s method doubles the number of significant figures after every iteration; however, we lose robustness if guess is poor

If \( f'(x_{\text{guess}}) \approx 0 \) -- doesn’t work

![Graph showing function and its derivative](image)

**Figure 6. NO intersection**

Another drawback is one needs a derivative of the function.

**Secant Method**

same as Newton’s, but uses \( f'(x) \) approximate

\[
f_{\text{approx}}(x) = f(x^{[k]}) - f(x^{[k-1]})/x^{[k]} - x^{[k-1]}
\]

*Bi-section method* works only for 1D problems, but *Newton/Secant* can be used for problems with greater dimension
**Broyden's Method (Multi-dimensional)**

\[ F(x) = F(x_0) + J(x_0) \cdot (x - x_0) \]

Method breaks down when \( J \) is singular

\[ \sum_j \left( \frac{\partial f_i}{\partial x_j} \right)_{x_0} (x_j - x_{0,j}) \]

\( f(x) = 0 \)

approx \( J = B \)

outer product is opposite of dot product

\[ B^{[k+1]} = B^{[k]} + \frac{F(x^{[k+1]}) \cdot (x^{[k+1]} - x^{[k]})^T}{\| \Delta x \|^2} \]

Outer Product:
\[
\begin{pmatrix}
F_1 \Delta x_1 & F_1 \Delta x_2 & F_1 \Delta x_3 & \cdots \\
F_2 \Delta x_1 & F_2 \Delta x_2 & F_2 \Delta x_3 & \cdots
\end{pmatrix}
\]

**Newton's Method (Multi-dimensional)**

\[ O = F(x_0) + J(x_0) \cdot (x - x_0) \]

\[ \frac{1}{LU} \Delta x = -F(x_0) \]

\[ \frac{1}{LU} B^{[k]} \Delta x = -F \]

\[ LU = LU^{[k+1]} \] without redoing factorization

Done in detail in homework problem.