MATH 350: Introduction to Computational Mathematics

Chapter IV: Locating Roots of Equations

Greg Fasshauer

Department of Applied Mathematics
Illinois Institute of Technology

Spring 2011

Outline

1. Motivation and Applications
2. Bisection
3. Newton's Method
4. Secant Method
5. Inverse Quadratic Interpolation
6. Root Finding in MATLAB: The Function `fzero`
7. Newton's Method for Systems of Nonlinear Equations
8. Optimization

Outline

fasshauer@iit.edu

We studied systems of linear equations in Chapter 2, and convinced ourselves of the importance for doing this.

We studied systems of linear equations in Chapter 2, and convinced ourselves of the importance for doing this.
Many real-life phenomena are more accurately described by nonlinear models. Thus, we often find ourselves asking:

We studied systems of linear equations in Chapter 2, and convinced ourselves of the importance for doing this.

Many real-life phenomena are more accurately described by nonlinear models. Thus, we often find ourselves asking:

Question

For what value(s) of x is the equation $f(x) = 0$ satisfied.

We studied systems of linear equations in Chapter 2, and convinced ourselves of the importance for doing this.
Many real-life phenomena are more accurately described by nonlinear models. Thus, we often find ourselves asking:

Question

For what value(s) of x is the equation $f(x) = 0$ satisfied.

Remark

Such an x is called a root (or zero) of the nonlinear equation $f(x) = 0$.

We studied systems of linear equations in Chapter 2, and convinced ourselves of the importance for doing this.

Many real-life phenomena are more accurately described by nonlinear models. Thus, we often find ourselves asking:

Question

For what value(s) of x is the equation $f(x) = 0$ satisfied.

Remark

Such an x is called a root (or zero) of the nonlinear equation $f(x) = 0$.

Example

Find the first positive root of the Bessel function

$$J_0(x) = \sum_{k=0}^{\infty} \frac{(-1)^k}{2^{2k}(k!)^2} x^{2k}.$$

We studied systems of linear equations in Chapter 2, and convinced ourselves of the importance for doing this.
Many real-life phenomena are more accurately described by nonlinear models. Thus, we often find ourselves asking:

Question

For what value(s) of x is the equation $f(x) = 0$ satisfied.

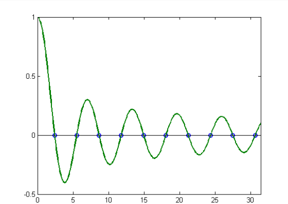
Remark

Such an x is called a root (or zero) of the nonlinear equation $f(x) = 0$.

Example

Find the first positive root of the Bessel function

$$J_0(x) = \sum_{k=0}^{\infty} \frac{(-1)^k}{2^{2k}(k!)^2} x^{2k}.$$



A more complicated example arises when the function f is given only indirectly as the solution of a differential equation.

Example

Consider the skydive model of Chapter 1. We can use a numerical method to find the velocity at any time $t \geq 0$. At what time will the skydiver hit the ground?

A more complicated example arises when the function f is given only indirectly as the solution of a differential equation.

Example

Consider the skydive model of Chapter 1. We can use a numerical method to find the velocity at any time $t \geq 0$. At what time will the skydiver hit the ground?

Solution

- First we need to find the position (altitude) for any time t from the initial position and calculated velocity (essentially the solution of another differential equation).
- Then we need to find the root of the position function — a rather complex procedure.

Most of this chapter will be concerned with the solution of a single nonlinear equation. However, systems of nonlinear equations are also important (and difficult) to solve.

Most of this chapter will be concerned with the solution of a single nonlinear equation. However, systems of nonlinear equations are also important (and difficult) to solve.

Example

Consider a missile M following the parametrized path

$$x_M(t) = t, \qquad y_M(t) = 1 - e^{-t},$$

and a missile interceptor I whose launch angle α we want to determine so that it will intersect the missile's path. Let the parametrized path for the interceptor be given as

$$x_I(t) = 1 - t\cos\alpha, \qquad y_I(t) = t\sin\alpha - \frac{t^2}{10}.$$

Most of this chapter will be concerned with the solution of a single nonlinear equation. However, systems of nonlinear equations are also important (and difficult) to solve.

Example

Consider a missile M following the parametrized path

$$x_M(t) = t, \qquad y_M(t) = 1 - e^{-t},$$

and a missile interceptor I whose launch angle α we want to determine so that it will intersect the missile's path. Let the parametrized path for the interceptor be given as

$$x_I(t) = 1 - t\cos\alpha, \qquad y_I(t) = t\sin\alpha - \frac{t^2}{10}.$$

Thus, we want to solve the nonlinear system

$$\left\{\begin{array}{rcl} t & = & 1 - t\cos\alpha \\ 1 - e^{-t} & = & t\sin\alpha - \frac{t^2}{10} \end{array}\right. \quad \text{or} \quad \left\{\begin{array}{rcl} f(t,\alpha) & = & t - 1 + t\cos\alpha = 0 \\ g(t,\alpha) & = & 1 - e^{-t} - t\sin\alpha + \frac{t^2}{10} = 0. \end{array}\right.$$

Outline

1. Motivation and Applications
2. **Bisection**
3. Newton's Method
4. Secant Method
5. Inverse Quadratic Interpolation
6. Root Finding in MATLAB: The Function `fzero`
7. Newton's Method for Systems of Nonlinear Equations
8. Optimization

Theorem (Intermediate Value Theorem)

If f is continuous on an interval $[a, b]$ and $f(a)$ and $f(b)$ are of opposite sign, then f has at least one root in $[a, b]$.

Theorem (Intermediate Value Theorem)

If f is continuous on an interval $[a, b]$ and $f(a)$ and $f(b)$ are of opposite sign, then f has at least one root in $[a, b]$.

This theorem provides the basis for a fool-proof — but rather slow — trial-and-error algorithm for finding a root of f:

Theorem (Intermediate Value Theorem)

If f is continuous on an interval $[a, b]$ and $f(a)$ and $f(b)$ are of opposite sign, then f has at least one root in $[a, b]$.

This theorem provides the basis for a fool-proof — but rather slow — trial-and-error algorithm for finding a root of f:

- Take the midpoint x of the interval $[a, b]$.
- If $f(x) = 0$ we're done.
- If not

Theorem (Intermediate Value Theorem)

If f is continuous on an interval $[a, b]$ and $f(a)$ and $f(b)$ are of opposite sign, then f has at least one root in $[a, b]$.

This theorem provides the basis for a fool-proof — but rather slow — trial-and-error algorithm for finding a root of f:

- Take the midpoint x of the interval $[a, b]$.
- If $f(x) = 0$ we're done.
- If not
  - Repeat entire procedure with either $[a, b] = [a, x]$ or $[a, b] = [x, b]$ (making sure that $f(a)$ and $f(b)$ have opposite signs).

Bisection Algorithm

```
while abs(b-a) > eps*abs(b)
    x = (a + b)/2;
    if sign(f(x)) == sign(f(b))
        b = x;     % set [a,x] as new [a,b]
    else
        a = x;     % set [x,b] as new [a,b]
    end
end
```

Bisection Algorithm

```
while abs(b-a) > eps*abs(b)
    x = (a + b)/2;
    if sign(f(x)) == sign(f(b))
        b = x;    % set [a,x] as new [a,b]
    else
        a = x;    % set [x,b] as new [a,b]
    end
end
```

The termination condition `while abs(b-a) > eps*abs(b)` ensures that the search continues until the root is found to within machine accuracy `eps`.

Bisection Algorithm

```
while abs(b-a) > eps*abs(b)
    x = (a + b)/2;
    if sign(f(x)) == sign(f(b))
        b = x;      % set [a,x] as new [a,b]
    else
        a = x;      % set [x,b] as new [a,b]
    end
end
```

The termination condition `while abs(b-a) > eps*abs(b)` ensures that the search continues until the root is found to within machine accuracy `eps`.

See `BisectDemo.m` and `bisect.m` for an illustration.

Bisection Algorithm

```
while abs(b-a) > eps*abs(b)
    x = (a + b)/2;
    if sign(f(x)) == sign(f(b))
        b = x;     % set [a,x] as new [a,b]
    else
        a = x;     % set [x,b] as new [a,b]
    end
end
```

The termination condition `while abs(b-a) > eps*abs(b)` ensures that the search continues until the root is found to within machine accuracy `eps`.

See `BisectDemo.m` and `bisect.m` for an illustration.

Remark

The algorithm as coded above should always — independent of f — converge in 52 iterations since the IEEE standard uses 52 bits for the mantissa, and we compute the answer with 1 bit accuracy.

Outline

By Taylor's theorem (assuming $f''(\xi)$ exists) we have

$$f(x) = f(x_0) + (x - x_0)f'(x_0) + \frac{(x - x_0)^2}{2} f''(\xi).$$

By Taylor's theorem (assuming $f''(\xi)$ exists) we have

$$f(x) = f(x_0) + (x - x_0)f'(x_0) + \frac{(x - x_0)^2}{2}f''(\xi).$$

So, for values of x_0 reasonably close to x we can approximate

$$f(x) \approx f(x_0) + (x - x_0)f'(x_0).$$

By Taylor's theorem (assuming $f''(\xi)$ exists) we have

$$f(x) = f(x_0) + (x - x_0)f'(x_0) + \frac{(x - x_0)^2}{2} f''(\xi).$$

So, for values of x_0 reasonably close to x we can approximate

$$f(x) \approx f(x_0) + (x - x_0)f'(x_0).$$

Since we are trying to find a root of f, i.e., we are hoping that $f(x) = 0$, we have

$$0 \approx f(x_0) + (x - x_0)f'(x_0) \quad \iff \quad x - x_0 \approx -\frac{f(x_0)}{f'(x_0)}.$$

By Taylor's theorem (assuming $f''(\xi)$ exists) we have

$$f(x) = f(x_0) + (x - x_0)f'(x_0) + \frac{(x - x_0)^2}{2} f''(\xi).$$

So, for values of x_0 reasonably close to x we can approximate

$$f(x) \approx f(x_0) + (x - x_0)f'(x_0).$$

Since we are trying to find a root of f, i.e., we are hoping that $f(x) = 0$, we have

$$0 \approx f(x_0) + (x - x_0)f'(x_0) \quad \Longleftrightarrow \quad x - x_0 \approx -\frac{f(x_0)}{f'(x_0)}.$$

This motivates the Newton iteration formula

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}, \qquad n = 0, 1, \ldots,$$

where an initial guess x_0 is required to start the iteration.

Graphical Interpretation

Consider the tangent line to the graph of f at x_n:

$$y - f(x_n) = f'(x_n)(x - x_n) \quad \Longrightarrow \quad y = f(x_n) + (x - x_n)f'(x_n).$$

Graphical Interpretation

Consider the tangent line to the graph of f at x_n:

$$y - f(x_n) = f'(x_n)(x - x_n) \quad \Longrightarrow \quad y = f(x_n) + (x - x_n)f'(x_n).$$

To see how this relates to Newton's method, set $y = 0$ and solve for x:

$$0 = f(x_n) + (x - x_n)f'(x_n) \quad \Longleftrightarrow \quad x = x_n - \frac{f(x_n)}{f'(x_n)}.$$

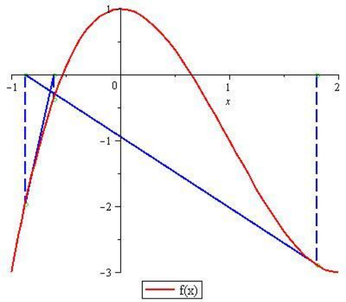
Graphical Interpretation

Consider the tangent line to the graph of f at x_n:

$$y - f(x_n) = f'(x_n)(x - x_n) \quad \Longrightarrow \quad y = f(x_n) + (x - x_n)f'(x_n).$$

To see how this relates to Newton's method, set $y = 0$ and solve for x:

$$0 = f(x_n) + (x - x_n)f'(x_n) \quad \Longleftrightarrow \quad x = x_n - \frac{f(x_n)}{f'(x_n)}.$$



Newton Iteration

```
while abs(x - xprev) > eps*abs(x)
    xprev = x;
    x = x - f(x)/fprime(x);
end
```

Newton Iteration

```
while abs(x - xprev) > eps*abs(x)
    xprev = x;
    x = x - f(x)/fprime(x);
end
```

See `NewtonDemo.m` and `newton.m` for an illustration. The Maple file `NewtonDemo.mw` contains an animated graphical illustration of the algorithm.

Newton Iteration

```
while abs(x - xprev) > eps*abs(x)
    xprev = x;
    x = x - f(x)/fprime(x);
end
```

See `NewtonDemo.m` and `newton.m` for an illustration. The Maple file `NewtonDemo.mw` contains an animated graphical illustration of the algorithm.

Remark

Convergence of Newton's method depends quite a bit on the choice of the initial guess x_0*. If successful, the algorithm above converges very quickly to within machine accuracy.*

Problem

How quickly does Newton's method converge? How fast does the error decrease from one iteration to the next?

Problem

How quickly does Newton's method converge? How fast does the error decrease from one iteration to the next?

Solution

Let's assume $f''(x)$ exists and $f'(x) \neq 0$ for all x of interest.

- Denote the root of f by x_*,
- and the error in iteration n by $e_n = x_n - x_*$.

Problem

How quickly does Newton's method converge? How fast does the error decrease from one iteration to the next?

Solution

Let's assume $f''(x)$ exists and $f'(x) \neq 0$ for all x of interest.

- Denote the root of f by x_*,
- and the error in iteration n by $e_n = x_n - x_*$.

Then

$$e_{n+1} = x_{n+1} - x_*$$

Problem

How quickly does Newton's method converge? How fast does the error decrease from one iteration to the next?

Solution

Let's assume $f''(x)$ exists and $f'(x) \neq 0$ for all x of interest.

- Denote the root of f by x_*,
- and the error in iteration n by $e_n = x_n - x_*$.

Then

$$\begin{aligned} e_{n+1} &= x_{n+1} - x_* \\ &= x_n - \frac{f(x_n)}{f'(x_n)} - x_* \end{aligned}$$

Problem

How quickly does Newton's method converge? How fast does the error decrease from one iteration to the next?

Solution

Let's assume $f''(x)$ exists and $f'(x) \neq 0$ for all x of interest.

- Denote the root of f by x_*,
- and the error in iteration n by $e_n = x_n - x_*$.

Then

$$\begin{aligned} e_{n+1} &= x_{n+1} - x_* \\ &= x_n - \frac{f(x_n)}{f'(x_n)} - x_* \\ &= e_n - \frac{f(x_n)}{f'(x_n)} \end{aligned}$$

Problem

How quickly does Newton's method converge? How fast does the error decrease from one iteration to the next?

Solution

Let's assume $f''(x)$ exists and $f'(x) \neq 0$ for all x of interest.

- Denote the root of f by x_*,
- and the error in iteration n by $e_n = x_n - x_*$.

Then

$$\begin{aligned} e_{n+1} &= x_{n+1} - x_* \\ &= x_n - \frac{f(x_n)}{f'(x_n)} - x_* \\ &= e_n - \frac{f(x_n)}{f'(x_n)} \\ &= \frac{e_n f'(x_n) - f(x_n)}{f'(x_n)} \end{aligned} \tag{1}$$

Solution (cont.)

On the other hand, a Taylor expansion gives

$$f(x_*) = f(x_n - e_n)$$

Solution (cont.)

On the other hand, a Taylor expansion gives

$$f(x_*) = f(x_n \underbrace{-e_n}_{=h})$$

Solution (cont.)

On the other hand, a Taylor expansion gives

$$f(x_*) = f(x_n \underbrace{-e_n}_{=h}) = f(x_n) - e_n f'(x_n) + \frac{e_n^2}{2} f''(\xi)$$

Solution (cont.)

On the other hand, a Taylor expansion gives

$$0 = f(x_*) = f(x_n \underbrace{-e_n}_{=h}) = f(x_n) - e_n f'(x_n) + \frac{e_n^2}{2} f''(\xi)$$

Solution (cont.)

On the other hand, a Taylor expansion gives

$$0 = f(x_*) = f(x_n \underbrace{-e_n}_{=h}) = f(x_n) - e_n f'(x_n) + \frac{e_n^2}{2} f''(\xi)$$

Rearrange:

$$e_n f'(x_n) - f(x_n) = \frac{e_n^2}{2} f''(\xi) \tag{2}$$

Solution (cont.)

On the other hand, a Taylor expansion gives

$$0 = f(x_*) = f(x_n \underbrace{-e_n}_{=h}) = f(x_n) - e_n f'(x_n) + \frac{e_n^2}{2} f''(\xi)$$

Rearrange:

$$e_n f'(x_n) - f(x_n) = \frac{e_n^2}{2} f''(\xi) \tag{2}$$

(2) in (1):

$$e_{n+1} = \frac{\frac{e_n^2}{2} f''(\xi)}{f'(x_n)}.$$

Solution (cont.)

On the other hand, a Taylor expansion gives

$$0 = f(x_*) = f(x_n \underbrace{-e_n}_{=h}) = f(x_n) - e_n f'(x_n) + \frac{e_n^2}{2} f''(\xi)$$

Rearrange:

$$e_n f'(x_n) - f(x_n) = \frac{e_n^2}{2} f''(\xi) \tag{2}$$

(2) in (1):

$$e_{n+1} = \frac{\frac{e_n^2}{2} f''(\xi)}{f'(x_n)}.$$

If x_n is close enough to x_* (so that also ξ is close to x_*) we have

$$e_{n+1} \approx \frac{f''(x_*)}{2f'(x_*)} e_n^2 \quad \Longrightarrow \quad e_{n+1} = \mathcal{O}(e_n^2).$$

Solution (cont.)

On the other hand, a Taylor expansion gives

$$0 = f(x_*) = f(x_n \underbrace{- e_n}_{=h}) = f(x_n) - e_n f'(x_n) + \frac{e_n^2}{2} f''(\xi)$$

Rearrange:

$$e_n f'(x_n) - f(x_n) = \frac{e_n^2}{2} f''(\xi) \tag{2}$$

(2) in (1):

$$e_{n+1} = \frac{\frac{e_n^2}{2} f''(\xi)}{f'(x_n)}.$$

If x_n is close enough to x_* (so that also ξ is close to x_*) we have

$$e_{n+1} \approx \frac{f''(x_*)}{2f'(x_*)} e_n^2 \quad \Longrightarrow \quad e_{n+1} = \mathcal{O}(e_n^2).$$

This is known as quadratic convergence, and implies that the number of correct digits approximately doubles in each iteration.

Outline

Problem

A significant drawback of Newton's method is its need for $f'(x_n)$.

Problem

A significant drawback of Newton's method is its need for $f'(x_n)$.

Solution

We approximate the value of the derivative $f'(x_n)$ by the slope s_n given as

$$s_n = \frac{f(x_n) - f(x_{n-1})}{x_n - x_{n-1}}.$$

Problem

A significant drawback of Newton's method is its need for $f'(x_n)$.

Solution

We approximate the value of the derivative $f'(x_n)$ by the slope s_n given as

$$s_n = \frac{f(x_n) - f(x_{n-1})}{x_n - x_{n-1}}.$$

Then we get the iteration formula

$$x_{n+1} = x_n - \frac{f(x_n)}{s_n}, \qquad n = 1, 2, \ldots.$$

Since s_n is the slope of the secant line from $(x_{n-1}, f(x_{n-1}))$ to $(x_n, f(x_n))$ this method is called the secant method.

Problem

A significant drawback of Newton's method is its need for $f'(x_n)$.

Solution

We approximate the value of the derivative $f'(x_n)$ by the slope s_n given as

$$s_n = \frac{f(x_n) - f(x_{n-1})}{x_n - x_{n-1}}.$$

Then we get the iteration formula

$$x_{n+1} = x_n - \frac{f(x_n)}{s_n}, \qquad n = 1, 2, \ldots.$$

Since s_n is the slope of the secant line from $(x_{n-1}, f(x_{n-1}))$ to $(x_n, f(x_n))$ this method is called the secant method.

Remark

The secant method requires two initial guesses, x_0 and x_1.

Secant Method

```
while abs(b-a) > eps*abs(b)
    c = a;
    a = b;
    b = b + (b - c)/(f(c)/f(b)-1);
end
```

Secant Method

```
while abs(b-a) > eps*abs(b)
    c = a;
    a = b;
    b = b + (b - c)/(f(c)/f(b)-1);
end
```

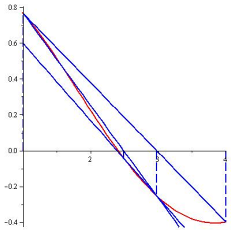
Note that $\frac{x_n - x_{n-1}}{\frac{f(x_{n-1})}{f(x_n)} - 1} = \frac{x_n - x_{n-1}}{\frac{f(x_{n-1}) - f(x_n)}{f(x_n)}} = \frac{(x_n - x_{n-1}) f(x_n)}{f(x_{n-1}) - f(x_n)} = \frac{f(x_n)}{s_n}$

Secant Method

```
while abs(b-a) > eps*abs(b)
    c = a;
    a = b;
    b = b + (b - c)/(f(c)/f(b)-1);
end
```

Note that $\frac{x_n - x_{n-1}}{\frac{f(x_{n-1})}{f(x_n)} - 1} = \frac{x_n - x_{n-1}}{\frac{f(x_{n-1}) - f(x_n)}{f(x_n)}} = \frac{(x_n - x_{n-1})f(x_n)}{f(x_{n-1}) - f(x_n)} = \frac{f(x_n)}{s_n}$

See `SecantDemo.m` and `secant.m` for an illustration. The Maple file `SecantDemo.mws` contains an animated graphical illustration of the algorithm.

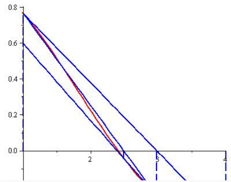


Secant Method

```
while abs(b-a) > eps*abs(b)
    c = a;
    a = b;
    b = b + (b - c)/(f(c)/f(b)-1);
end
```

Note that $\frac{x_n-x_{n-1}}{\frac{f(x_{n-1})}{f(x_n)}-1} = \frac{x_n-x_{n-1}}{\frac{f(x_{n-1})-f(x_n)}{f(x_n)}} = \frac{(x_n-x_{n-1})f(x_n)}{f(x_{n-1})-f(x_n)} = \frac{f(x_n)}{s_n}$

See `SecantDemo.m` and `secant.m` for an illustration. The Maple file `SecantDemo.mws` contains an animated graphical illustration of the algorithm.



Remark

Convergence of the secant method also depends on the choice of initial guesses. If successful, the algorithm converges superlinearly, i.e., $e_{n+1} = \mathcal{O}(e_n^{\phi})$, *where* $\phi = (\sqrt{5}+1)/2$, *the golden ratio.*

Outline

We can interpret the secant method as using the linear interpolant to the data $(x_{n-1}, f(x_{n-1})), (x_n, f(x_n))$ to approximate the zero of the function f.

We can interpret the secant method as using the linear interpolant to the data $(x_{n-1}, f(x_{n-1})), (x_n, f(x_n))$ to approximate the zero of the function f.

Question

Wouldn't it be better (if possible) to use a quadratic interpolant to three data points to get this job done?

We can interpret the secant method as using the linear interpolant to the data $(x_{n-1}, f(x_{n-1})), (x_n, f(x_n))$ to approximate the zero of the function f.

Question

Wouldn't it be better (if possible) to use a quadratic interpolant to three data points to get this job done?

Answer

In principle, "yes". The resulting method is called inverse quadratic interpolation (IQI).

> *IQI is like an immature race horse. It moves very quickly when it is near the finish line, but its global behavior can be erratic [NCM].*

How does inverse quadratic interpolation work?

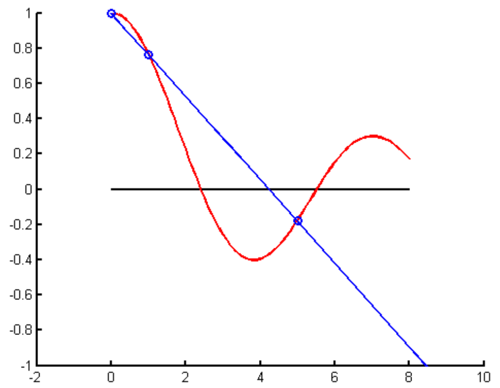
Assume we have 3 data points: $(a, f(a)), (b, f(b)), (c, f(c))$.

How does inverse quadratic interpolation work?

Assume we have 3 data points: $(a, f(a)), (b, f(b)), (c, f(c))$.
Instead of interpolating the data directly with a quadratic polynomial we reverse the roles of x and y since then we can evaluate the resulting polynomial at $y = 0$; and this gives an approximation to the root of f!

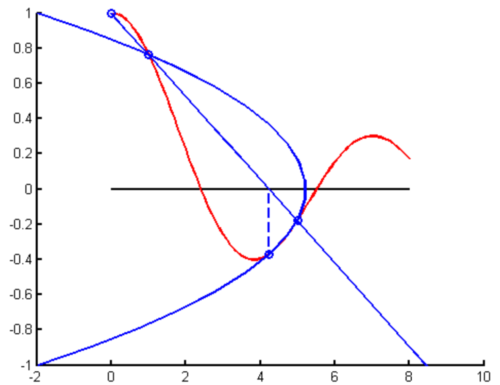
How does inverse quadratic interpolation work?

Assume we have 3 data points: $(a, f(a)), (b, f(b)), (c, f(c))$.
Instead of interpolating the data directly with a quadratic polynomial we reverse the roles of x and y since then we can evaluate the resulting polynomial at $y = 0$; and this gives an approximation to the root of f!



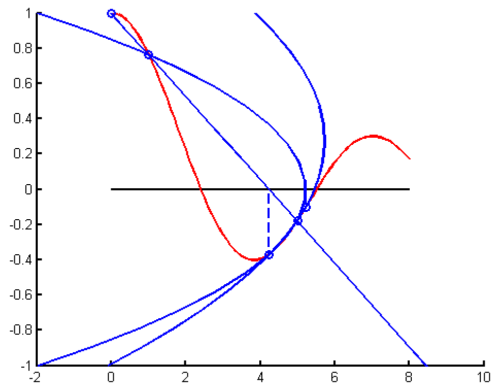
How does inverse quadratic interpolation work?

Assume we have 3 data points: $(a, f(a)), (b, f(b)), (c, f(c))$.
Instead of interpolating the data directly with a quadratic polynomial we reverse the roles of x and y since then we can evaluate the resulting polynomial at $y = 0$; and this gives an approximation to the root of f!



How does inverse quadratic interpolation work?

Assume we have 3 data points: $(a, f(a)), (b, f(b)), (c, f(c))$.
Instead of interpolating the data directly with a quadratic polynomial we reverse the roles of x and y since then we can evaluate the resulting polynomial at $y = 0$; and this gives an approximation to the root of f!



IQI Method

```
while abs(c-b) > eps*abs(c)
    x = polyinterp([f(a),f(b),f(c)],[a,b,c],0);
    a = b;
    b = c;
    c = x;
end
```

IQI Method

```
while abs(c-b) > eps*abs(c)
    x = polyinterp([f(a),f(b),f(c)],[a,b,c],0);
    a = b;
    b = c;
    c = x;
end
```

See the MATLAB script `IQIDemo.m` which calls the function `iqi.m`.

IQI Method

```
while abs(c-b) > eps*abs(c)
    x = polyinterp([f(a),f(b),f(c)],[a,b,c],0);
    a = b;
    b = c;
    c = x;
end
```

See the MATLAB script `IQIDemo.m` which calls the function `iqi.m`.

Remark

One of the major challenges for the IQI method is to ensure that the function values, i.e., $f(a)$, $f(b)$ and $f(c)$, are distinct — since we are using them as our interpolation nodes.

Outline

1. Motivation and Applications
2. Bisection
3. Newton's Method
4. Secant Method
5. Inverse Quadratic Interpolation
6. **Root Finding in MATLAB: The Function `fzero`**
7. Newton's Method for Systems of Nonlinear Equations
8. Optimization

The MATLAB code `fzerotx.m` from [NCM] is based on a combination of three of the methods discussed above: bisection, secant, and IQI.

The MATLAB code `fzerotx.m` from [NCM] is based on a combination of three of the methods discussed above: bisection, secant, and IQI.

- Start with a and b so that $f(a)$ and $f(b)$ have opposite signs.
- Use a secant step to give c between a and b.
- Repeat the following steps until $|b - a| < \epsilon|b|$ or $f(b) = 0$.
- Arrange a, b, and c so that
  - $f(a)$ and $f(b)$ have opposite signs,
  - $|f(b)| \leq |f(a)|$,
  - c is the previous value of b.
- If $c \neq a$, consider an IQI step.
- If $c = a$, consider a secant step.
- If the IQI or secant step is in the interval $[a, b]$, take it.
- If the step is not in the interval, use bisection.

The MATLAB code `fzerotx.m` from [NCM] is based on a combination of three of the methods discussed above: bisection, secant, and IQI.

- Start with a and b so that $f(a)$ and $f(b)$ have opposite signs.
- Use a secant step to give c between a and b.
- Repeat the following steps until $|b - a| < \epsilon|b|$ or $f(b) = 0$.
- Arrange a, b, and c so that
  - $f(a)$ and $f(b)$ have opposite signs,
  - $|f(b)| \leq |f(a)|$,
  - c is the previous value of b.
- If $c \neq a$, consider an IQI step.
- If $c = a$, consider a secant step.
- If the IQI or secant step is in the interval $[a, b]$, take it.
- If the step is not in the interval, use bisection.

The algorithm always works and combines the robustness of the bisection method and the speed of the secant and IQI methods.

The MATLAB code `fzerotx.m` from [NCM] is based on a combination of three of the methods discussed above: bisection, secant, and IQI.

- Start with a and b so that $f(a)$ and $f(b)$ have opposite signs.
- Use a secant step to give c between a and b.
- Repeat the following steps until $|b - a| < \epsilon|b|$ or $f(b) = 0$.
- Arrange a, b, and c so that
  - $f(a)$ and $f(b)$ have opposite signs,
  - $|f(b)| \leq |f(a)|$,
  - c is the previous value of b.
- If $c \neq a$, consider an IQI step.
- If $c = a$, consider a secant step.
- If the IQI or secant step is in the interval $[a, b]$, take it.
- If the step is not in the interval, use bisection.

The algorithm always works and combines the robustness of the bisection method and the speed of the secant and IQI methods. This algorithm is also known as Brent's method.

Root finding in MATLAB (cont.)

A step-by-step exploration of the `fzero` algorithm is possible with `fzerogui.m` from [NCM].
To find the first positive root of J_0 use

```
fzerogui(@(x) besselj(0,x),[0,4]),
```

where `@(x) besselj(0,x)` is an *anonymous function* of the one variable `x` (while the argument `@besselj` would be a *function handle* for a function of two variables – and therefore confuse the routine `fzerogui`).

Outline

1. Motivation and Applications
2. Bisection
3. Newton's Method
4. Secant Method
5. Inverse Quadratic Interpolation
6. Root Finding in MATLAB: The Function `fzero`
7. Newton's Method for Systems of Nonlinear Equations
8. Optimization

Example

We now want to solve a nonlinear system such as

$$\begin{aligned} f(t,\alpha) &= t - 1 + t\cos\alpha = 0 \\ g(t,\alpha) &= 1 - e^{-t} - t\sin\alpha + \frac{t^2}{10} = 0. \end{aligned}$$

Example

We now want to solve a nonlinear system such as

$$\begin{aligned} f(t,\alpha) &= t - 1 + t\cos\alpha = 0 \\ g(t,\alpha) &= 1 - e^{-t} - t\sin\alpha + \tfrac{t^2}{10} = 0. \end{aligned}$$

Earlier we derived the basic Newton method from the truncated Taylor expansion (note that here I've changed the earlier notation of x_0 to c)

$$f(x) = f(c) + (x-c)f'(c) + \frac{(x-c)^2}{2}f''(\xi).$$

Then

$$f(x) \approx f(c) + (x-c)f'(c) \quad \overset{f(x)=0}{\Longleftrightarrow} \quad x \approx c - \frac{f(c)}{f'(c)}.$$

Example

We now want to solve a nonlinear system such as

$$\begin{aligned} f(t,\alpha) &= t - 1 + t\cos\alpha = 0 \\ g(t,\alpha) &= 1 - e^{-t} - t\sin\alpha + \tfrac{t^2}{10} = 0. \end{aligned}$$

Earlier we derived the basic Newton method from the truncated Taylor expansion (note that here I've changed the earlier notation of x_0 to c)

$$f(x) = f(c) + (x-c)f'(c) + \frac{(x-c)^2}{2}f''(\xi).$$

Then

$$f(x) \approx f(c) + (x-c)f'(c) \quad \overset{f(x)=0}{\Longleftrightarrow} \quad x \approx c - \frac{f(c)}{f'(c)}.$$

Using vector notation, our nonlinear system above can be written as

$$\boldsymbol{f}(\boldsymbol{x}) = \boldsymbol{0},$$

where $\boldsymbol{x} = [t,\alpha]^T$ and $\boldsymbol{f} = [f,g]^T$.

Example

We now want to solve a nonlinear system such as

$$\begin{aligned} f(t,\alpha) &= t - 1 + t\cos\alpha = 0 \\ g(t,\alpha) &= 1 - e^{-t} - t\sin\alpha + \tfrac{t^2}{10} = 0. \end{aligned}$$

Earlier we derived the basic Newton method from the truncated Taylor expansion (note that here I've changed the earlier notation of x_0 to c)

$$f(x) = f(c) + (x-c)f'(c) + \frac{(x-c)^2}{2} f''(\xi).$$

Then

$$f(x) \approx f(c) + (x-c)f'(c) \quad \overset{f(x)=0}{\Longleftrightarrow} \quad x \approx c - \frac{f(c)}{f'(c)}.$$

Using vector notation, our nonlinear system above can be written as

$$\boldsymbol{f}(\boldsymbol{x}) = \mathbf{0},$$

where $\boldsymbol{x} = [t,\alpha]^T$ and $\boldsymbol{f} = [f,g]^T$.
We therefore need a multivariate version of Newton's method.

For a single function f of m variables we would need the expansion

$$f(\boldsymbol{x}) = f(\boldsymbol{c}) + ((\boldsymbol{x}-\boldsymbol{c})^T\nabla)f(\boldsymbol{c}) + \frac{1}{2}((\boldsymbol{x}-\boldsymbol{c})^T\nabla)^2 f(\xi),$$

where $\nabla = \left[\frac{\partial}{\partial x_1}, \frac{\partial}{\partial x_2}, \ldots, \frac{\partial}{\partial x_m}\right]^T$ is the gradient operator.

For a single function f of m variables we would need the expansion

$$f(\boldsymbol{x}) = f(\boldsymbol{c}) + ((\boldsymbol{x}-\boldsymbol{c})^T\nabla)f(\boldsymbol{c}) + \frac{1}{2}((\boldsymbol{x}-\boldsymbol{c})^T\nabla)^2 f(\xi),$$

where $\nabla = \left[\frac{\partial}{\partial x_1}, \frac{\partial}{\partial x_2}, \ldots, \frac{\partial}{\partial x_m}\right]^T$ is the gradient operator.

Example

If we have only $m = 2$ variables, i.e., $\boldsymbol{x} = [x_1, x_2]^T$, this becomes

$$\begin{aligned} f(x_1,x_2) &= f(c_1,c_2) + \left((x_1-c_1)\frac{\partial}{\partial x_1} + (x_2-c_2)\frac{\partial}{\partial x_2}\right) f(c_1,c_2) \\ &\quad + \frac{1}{2}\left((x_1-c_1)\frac{\partial}{\partial x_1} + (x_2-c_2)\frac{\partial}{\partial x_2}\right)^2 f(\xi_1,\xi_2) \end{aligned}$$

fasshauer@iit.edu

For a single function f of m variables we would need the expansion

$$f(\boldsymbol{x}) = f(\boldsymbol{c}) + ((\boldsymbol{x}-\boldsymbol{c})^T\nabla)f(\boldsymbol{c}) + \frac{1}{2}((\boldsymbol{x}-\boldsymbol{c})^T\nabla)^2 f(\boldsymbol{\xi}),$$

where $\nabla = \left[\frac{\partial}{\partial x_1}, \frac{\partial}{\partial x_2}, \ldots, \frac{\partial}{\partial x_m}\right]^T$ is the gradient operator.

Example

If we have only $m = 2$ variables, i.e., $\boldsymbol{x} = [x_1, x_2]^T$, this becomes

$$\begin{aligned} f(x_1,x_2) &= f(c_1,c_2) + \left((x_1-c_1)\frac{\partial}{\partial x_1} + (x_2-c_2)\frac{\partial}{\partial x_2}\right) f(c_1,c_2) \\ &\quad + \frac{1}{2}\left((x_1-c_1)\frac{\partial}{\partial x_1} + (x_2-c_2)\frac{\partial}{\partial x_2}\right)^2 f(\xi_1,\xi_2) \\ &= f(c_1,c_2) + (x_1-c_1)\frac{\partial f}{\partial x_1}(c_1,c_2) + (x_2-c_2)\frac{\partial f}{\partial x_2}(c_1,c_2) \\ &\quad + \left(\frac{(x_1-c_1)^2}{2}\frac{\partial^2}{\partial x_1^2} + (x_1-c_1)(x_2-c_2)\frac{\partial^2}{\partial x_1 \partial x_2} + \frac{(x_2-c_2)^2}{2}\frac{\partial^2}{\partial x_2^2}\right) f(\xi_1,\xi_2). \end{aligned}$$

Example (cont.)

Therefore, we can approximate f by

$$f(x_1, x_2) \approx f(c_1, c_2) + (x_1 - c_1)\frac{\partial f}{\partial x_1}(c_1, c_2) + (x_2 - c_2)\frac{\partial f}{\partial x_2}(c_1, c_2)$$

Example (cont.)

Therefore, we can approximate f by

$$f(x_1, x_2) \approx f(c_1, c_2) + (x_1 - c_1)\frac{\partial f}{\partial x_1}(c_1, c_2) + (x_2 - c_2)\frac{\partial f}{\partial x_2}(c_1, c_2)$$

Back to more compact operator notation we have

$$f(\boldsymbol{x}) \approx f(\boldsymbol{c}) + ((\boldsymbol{x} - \boldsymbol{c})^T \nabla) f(\boldsymbol{c}).$$

Example (cont.)

Therefore, we can approximate f by

$$f(x_1, x_2) \approx f(c_1, c_2) + (x_1 - c_1)\frac{\partial f}{\partial x_1}(c_1, c_2) + (x_2 - c_2)\frac{\partial f}{\partial x_2}(c_1, c_2)$$

Back to more compact operator notation we have

$$f(\boldsymbol{x}) \approx f(\boldsymbol{c}) + ((\boldsymbol{x} - \boldsymbol{c})^T \nabla) f(\boldsymbol{c}).$$

Note that this approximation is a linearization of f and in fact denotes the tangent plane to the graph of f at the point $\boldsymbol{c}$.

More generally, we have the multivariate Taylor expansion:

$$f(\boldsymbol{x}) = \sum_{k=0}^{n} \frac{1}{k!}((\boldsymbol{x}-\boldsymbol{c})^T\nabla)^k f(\boldsymbol{c}) + E_{n+1}(\boldsymbol{x}). \tag{3}$$

Here the remainder is

$$E_{n+1}(\boldsymbol{x}) = \frac{1}{(n+1)!}((\boldsymbol{x}-\boldsymbol{c})^T\nabla)^{n+1} f(\xi)$$

where $\xi = \boldsymbol{c} + \theta(\boldsymbol{x}-\boldsymbol{c})$ with $0 < \theta < 1$ a point somewhere on the line connecting $\boldsymbol{c}$ and $\boldsymbol{x}$, and $\nabla = \left[\frac{\partial}{\partial x_1}, \frac{\partial}{\partial x_2}, \ldots, \frac{\partial}{\partial x_m}\right]^T$ is the gradient operator as before.

More generally, we have the multivariate Taylor expansion:

$$f(\boldsymbol{x}) = \sum_{k=0}^{n} \frac{1}{k!}((\boldsymbol{x}-\boldsymbol{c})^T\nabla)^k f(\boldsymbol{c}) + E_{n+1}(\boldsymbol{x}). \tag{3}$$

Here the remainder is

$$E_{n+1}(\boldsymbol{x}) = \frac{1}{(n+1)!}((\boldsymbol{x}-\boldsymbol{c})^T\nabla)^{n+1} f(\xi)$$

where $\xi = \boldsymbol{c} + \theta(\boldsymbol{x}-\boldsymbol{c})$ with $0 < \theta < 1$ a point somewhere on the line connecting $\boldsymbol{c}$ and $\boldsymbol{x}$, and $\nabla = \left[\frac{\partial}{\partial x_1}, \frac{\partial}{\partial x_2}, \ldots, \frac{\partial}{\partial x_m}\right]^T$ is the gradient operator as before.

Remark

Note, however, that this slide is added as a reference/reminder only and is not required for the derivation of the multivariate Newton method.

Now we want to tackle the full problem, i.e., we want to solve the following (square) *system of nonlinear equations*:

$$\begin{aligned} f_1(x_1, x_2, \ldots, x_m) &= 0, \\ f_2(x_1, x_2, \ldots, x_m) &= 0, \\ &\vdots \\ f_m(x_1, x_2, \ldots, x_m) &= 0. \end{aligned} \tag{4}$$

Now we want to tackle the full problem, i.e., we want to solve the following (square) *system of nonlinear equations*:

$$\begin{aligned} f_1(x_1, x_2, \ldots, x_m) &= 0, \\ f_2(x_1, x_2, \ldots, x_m) &= 0, \\ &\vdots \\ f_m(x_1, x_2, \ldots, x_m) &= 0. \end{aligned} \tag{4}$$

To derive Newton's method for (4) we write it in the form,

$$f_i(\boldsymbol{x}) = 0, \qquad i = 1, \ldots, m.$$

By linearizing f_i, $i = 1, \ldots, m$, as discussed above we have

$$f_i(\boldsymbol{x}) \approx f_i(\boldsymbol{c}) + ((\boldsymbol{x} - \boldsymbol{c})^T \nabla) f_i(\boldsymbol{c}).$$

Since $f_i(\boldsymbol{x}) = 0$ we get

$$\begin{aligned} -f_i(\boldsymbol{c}) &\approx ((\boldsymbol{x}-\boldsymbol{c})^T\nabla)f_i(\boldsymbol{c}) \\ &= (x_1-c_1)\frac{\partial f_i}{\partial x_1}(\boldsymbol{c}) + (x_2-c_2)\frac{\partial f_i}{\partial x_2}(\boldsymbol{c}) + \ldots + (x_m-c_m)\frac{\partial f_i}{\partial x_m}(\boldsymbol{c}). \end{aligned}$$

Since $f_i(\boldsymbol{x}) = 0$ we get

$$\begin{aligned}
-f_i(\boldsymbol{c}) &\approx ((\boldsymbol{x}-\boldsymbol{c})^T\nabla)f_i(\boldsymbol{c}) \\
&= (x_1 - c_1)\frac{\partial f_i}{\partial x_1}(\boldsymbol{c}) + (x_2 - c_2)\frac{\partial f_i}{\partial x_2}(\boldsymbol{c}) + \ldots + (x_m - c_m)\frac{\partial f_i}{\partial x_m}(\boldsymbol{c}).
\end{aligned}$$

Therefore, we have a linear system for the unknown approximate root $\boldsymbol{x}$ of (4):

$$\begin{aligned}
-f_1(c_1,\ldots,c_m) &= (x_1 - c_1)\frac{\partial f_1}{\partial x_1}(c_1,\ldots,c_m) + \ldots + (x_m - c_m)\frac{\partial f_1}{\partial x_m}(c_1,\ldots,c_m), \\
-f_2(c_1,\ldots,c_m) &= (x_1 - c_1)\frac{\partial f_2}{\partial x_1}(c_1,\ldots,c_m) + \ldots + (x_m - c_m)\frac{\partial f_2}{\partial x_m}(c_1,\ldots,c_m), \\
&\vdots \\
-f_m(c_1,\ldots,c_m) &= (x_1 - c_1)\frac{\partial f_m}{\partial x_1}(c_1,\ldots,c_m) + \ldots + (x_m - c_m)\frac{\partial f_m}{\partial x_m}(c_1,\ldots,c_m).
\end{aligned} \tag{5}$$

To simplify notation a bit we now introduce $\boldsymbol{h} = [h_1, \ldots, h_m]^T = \boldsymbol{x} - \boldsymbol{c}$, and note that (5) is a linear system for $\boldsymbol{h}$ of the form

$$\mathsf{J}(\boldsymbol{c})\boldsymbol{h} = -\boldsymbol{f}(\boldsymbol{c}),$$

where $\boldsymbol{f} = [f_1, \ldots, f_m]^T$ and

$$\mathsf{J} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \cdots & \frac{\partial f_1}{\partial x_m} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \cdots & \frac{\partial f_2}{\partial x_m} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \frac{\partial f_m}{\partial x_2} & \cdots & \frac{\partial f_m}{\partial x_m} \end{bmatrix}$$

is called the *Jacobian* of $\boldsymbol{f}$.

To simplify notation a bit we now introduce $\boldsymbol{h} = [h_1, \ldots, h_m]^T = \boldsymbol{x} - \boldsymbol{c}$, and note that (5) is a linear system for $\boldsymbol{h}$ of the form

$$\mathsf{J}(\boldsymbol{c})\boldsymbol{h} = -\boldsymbol{f}(\boldsymbol{c}),$$

where $\boldsymbol{f} = [f_1, \ldots, f_m]^T$ and

$$\mathsf{J} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \cdots & \frac{\partial f_1}{\partial x_m} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \cdots & \frac{\partial f_2}{\partial x_m} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \frac{\partial f_m}{\partial x_2} & \cdots & \frac{\partial f_m}{\partial x_m} \end{bmatrix}$$

is called the *Jacobian* of $\boldsymbol{f}$.

Since $\boldsymbol{h} = \boldsymbol{x} - \boldsymbol{c}$ or $\boldsymbol{x} = \boldsymbol{c} + \boldsymbol{h}$ we see that $\boldsymbol{h}$ is an update to the previous approximation $\boldsymbol{c}$ of the root $\boldsymbol{x}$.

Algorithm

Newton's method for square nonlinear systems is performed by

```
Input $\boldsymbol{f}$, J, $\boldsymbol{x}^{(0)}$
for $n = 0, 1, 2, \ldots$ do
    Solve $\mathrm{J}(\boldsymbol{x}^{(n)})\boldsymbol{h} = -\boldsymbol{f}(\boldsymbol{x}^{(n)})$ for $\boldsymbol{h}$
    Update $\boldsymbol{x}^{(n+1)} = \boldsymbol{x}^{(n)} + \boldsymbol{h}$
end
Output $\boldsymbol{x}^{(n+1)}$
```

Algorithm

Newton's method for square nonlinear systems is performed by

```
Input $\boldsymbol{f}$, J, $\boldsymbol{x}^{(0)}$
for $n = 0, 1, 2, \ldots$ do
    Solve $J(\boldsymbol{x}^{(n)})\boldsymbol{h} = -\boldsymbol{f}(\boldsymbol{x}^{(n)})$ for $\boldsymbol{h}$
    Update $\boldsymbol{x}^{(n+1)} = \boldsymbol{x}^{(n)} + \boldsymbol{h}$
end
Output $\boldsymbol{x}^{(n+1)}$
```

Remark

If we symbolically write $\mathbf{f}'$ *instead of* J*, then the Newton iteration becomes*

$$\boldsymbol{x}^{(n+1)} = \boldsymbol{x}^{(n)} - \underbrace{\left[\boldsymbol{f}'(\boldsymbol{x}^{(n)})\right]^{-1}}_{\text{matrix}} \boldsymbol{f}(\boldsymbol{x}^{(n)}),$$

which looks just like the Newton iteration formula for the single equation/single variable case.

Example

Solve the missile intercept problem

$$\begin{aligned} t - 1 + t\cos\alpha &= 0 \\ 1 - e^{-t} - t\sin\alpha + \frac{t^2}{10} &= 0. \end{aligned}$$

Example

Solve the missile intercept problem

$$\begin{aligned} t - 1 + t\cos\alpha &= 0 \\ 1 - \mathrm{e}^{-t} - t\sin\alpha + \tfrac{t^2}{10} &= 0. \end{aligned}$$

Here

$$\boldsymbol{f}(t,\alpha) = \begin{bmatrix} f_1(t,\alpha) \\ f_2(t,\alpha) \end{bmatrix} = \begin{bmatrix} t - 1 + t\cos\alpha \\ 1 - \mathrm{e}^{-t} - t\sin\alpha + \tfrac{t^2}{10} \end{bmatrix}$$

and

$$\mathrm{J}(t,\alpha) = \begin{bmatrix} \frac{\partial f_1}{\partial t} & \frac{\partial f_1}{\partial \alpha} \\ \frac{\partial f_2}{\partial t} & \frac{\partial f_2}{\partial \alpha} \end{bmatrix}(t,\alpha) = \begin{bmatrix} 1+\cos(\alpha) & -t\sin(\alpha) \\ \mathrm{e}^{-t} - \sin(\alpha) + t/5 & -t\cos(\alpha) \end{bmatrix}.$$

Example

Solve the missile intercept problem

$$\begin{aligned} t - 1 + t\cos\alpha &= 0 \\ 1 - e^{-t} - t\sin\alpha + \frac{t^2}{10} &= 0. \end{aligned}$$

Here

$$\boldsymbol{f}(t,\alpha) = \begin{bmatrix} f_1(t,\alpha) \\ f_2(t,\alpha) \end{bmatrix} = \begin{bmatrix} t - 1 + t\cos\alpha \\ 1 - e^{-t} - t\sin\alpha + \frac{t^2}{10} \end{bmatrix}$$

and

$$\mathrm{J}(t,\alpha) = \begin{bmatrix} \frac{\partial f_1}{\partial t} & \frac{\partial f_1}{\partial \alpha} \\ \frac{\partial f_2}{\partial t} & \frac{\partial f_2}{\partial \alpha} \end{bmatrix}(t,\alpha) = \begin{bmatrix} 1 + \cos(\alpha) & -t\sin(\alpha) \\ e^{-t} - \sin(\alpha) + t/5 & -t\cos(\alpha) \end{bmatrix}.$$

This example is illustrated in the MATLAB script `NewtonmvDemo.m` which requires `newtonmv.m`, `missile_f.m` and `missile_j.m`.

Example

Solve

$$\begin{aligned} x^2 + y^2 &= 4 \\ xy &= 1, \end{aligned}$$

which corresponds to finding the intersection points of a circle and a hyperbola in the plane.

Example

Solve

$$\begin{aligned} x^2 + y^2 &= 4 \\ xy &= 1, \end{aligned}$$

which corresponds to finding the intersection points of a circle and a hyperbola in the plane. Here

$$\boldsymbol{f}(x, y) = \begin{bmatrix} f_1(x, y) \\ f_2(x, y) \end{bmatrix} = \begin{bmatrix} x^2 + y^2 - 4 \\ xy - 1 \end{bmatrix}$$

and

$$\mathrm{J}(x, y) = \begin{bmatrix} \frac{\partial f_1}{\partial x} & \frac{\partial f_1}{\partial y} \\ \frac{\partial f_2}{\partial x} & \frac{\partial f_2}{\partial y} \end{bmatrix}(x, y) = \begin{bmatrix} 2x & 2y \\ y & x \end{bmatrix}.$$

Example

Solve

$$\begin{aligned} x^2 + y^2 &= 4 \\ xy &= 1, \end{aligned}$$

which corresponds to finding the intersection points of a circle and a hyperbola in the plane. Here

$$\boldsymbol{f}(x, y) = \begin{bmatrix} f_1(x, y) \\ f_2(x, y) \end{bmatrix} = \begin{bmatrix} x^2 + y^2 - 4 \\ xy - 1 \end{bmatrix}$$

and

$$\mathrm{J}(x, y) = \begin{bmatrix} \frac{\partial f_1}{\partial x} & \frac{\partial f_1}{\partial y} \\ \frac{\partial f_2}{\partial x} & \frac{\partial f_2}{\partial y} \end{bmatrix}(x, y) = \begin{bmatrix} 2x & 2y \\ y & x \end{bmatrix}.$$

This example is also illustrated in the MATLAB script `NewtonmvDemo.m`. The files `circhyp_f.m` and `circhyp_j.m` are also needed.

Remark

1. *Newton's method requires the user to input the $m \times m$ Jacobian matrix (which depends on the specific nonlinear system to be solved). This is rather cumbersome.*

Remark

1. *Newton’s method requires the user to input the $m \times m$ Jacobian matrix (which depends on the specific nonlinear system to be solved). This is rather cumbersome.*
2. *In each iteration an $m \times m$ (dense) linear system has to be solved. This makes Newton’s method very expensive and slow.*

Remark

1. *Newton's method requires the user to input the $m \times m$ Jacobian matrix (which depends on the specific nonlinear system to be solved). This is rather cumbersome.*
2. *In each iteration an $m \times m$ (dense) linear system has to be solved. This makes Newton's method very expensive and slow.*
3. *For "good" starting values, Newton's method converges quadratically to simple zeros, i.e., solutions for which $\mathsf{J}^{-1}(\boldsymbol{z})$ exists.*

Remark

1. *Newton's method requires the user to input the $m \times m$ Jacobian matrix (which depends on the specific nonlinear system to be solved). This is rather cumbersome.*
2. *In each iteration an $m \times m$ (dense) linear system has to be solved. This makes Newton's method very expensive and slow.*
3. *For "good" starting values, Newton's method converges quadratically to simple zeros, i.e., solutions for which $\mathrm{J}^{-1}(\boldsymbol{z})$ exists.*
4. *Also, there is no built-in* MATLAB *code for nonlinear systems. However, the Optimization Toolbox (part of the student version) has a function* `fsolve` *that can be used for this purpose (note that it does not require the Jacobian of f). Try, e.g.,*
   `fsolve(@circhyp_f, [-3 1.5]).`

Remark

1. *Newton's method requires the user to input the $m \times m$ Jacobian matrix (which depends on the specific nonlinear system to be solved). This is rather cumbersome.*
2. *In each iteration an $m \times m$ (dense) linear system has to be solved. This makes Newton's method very expensive and slow.*
3. *For "good" starting values, Newton's method converges quadratically to simple zeros, i.e., solutions for which $\mathsf{J}^{-1}(\boldsymbol{z})$ exists.*
4. *Also, there is no built-in* MATLAB *code for nonlinear systems. However, the Optimization Toolbox (part of the student version) has a function* `fsolve` *that can be used for this purpose (note that it does not require the Jacobian of f). Try, e.g.,* `fsolve(@circhyp_f, [-3 1.5])`.
5. *More details for nonlinear systems are provided in MATH 477 and/or MATH 478.*

Outline

1. Motivation and Applications
2. Bisection
3. Newton's Method
4. Secant Method
5. Inverse Quadratic Interpolation
6. Root Finding in MATLAB: The Function `fzero`
7. Newton's Method for Systems of Nonlinear Equations
8. Optimization

A problem closely related to that of root finding is the need to find a maximum or minimum of a given function f.

A problem closely related to that of root finding is the need to find a maximum or minimum of a given function f.

For a continuous function of one variable this means that we need to find the critical points, i.e., the roots of the derivative of f.

A problem closely related to that of root finding is the need to find a maximum or minimum of a given function f.

For a continuous function of one variable this means that we need to find the critical points, i.e., the roots of the derivative of f.

Since we decided earlier that Newton's method (which requires knowledge of f') is in many cases too complicated and costly to use, we would again like to find a method which can find the minimum of f (or of $-f$ if we're interested in finding the maximum of f) on a given interval without requiring knowledge of f'.

A problem closely related to that of root finding is the need to find a maximum or minimum of a given function f.

For a continuous function of one variable this means that we need to find the critical points, i.e., the roots of the derivative of f.

Since we decided earlier that Newton's method (which requires knowledge of f') is in many cases too complicated and costly to use, we would again like to find a method which can find the minimum of f (or of $-f$ if we're interested in finding the maximum of f) on a given interval without requiring knowledge of f'.

The final MATLAB function will again be a robust hybrid method.

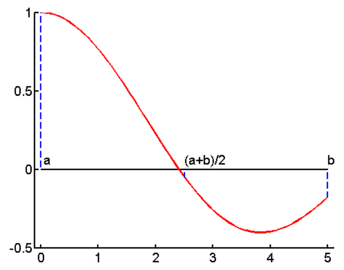
Problem

Use the bisection strategy to compute a minimum of f.

Problem

Use the bisection strategy to compute a minimum of f.

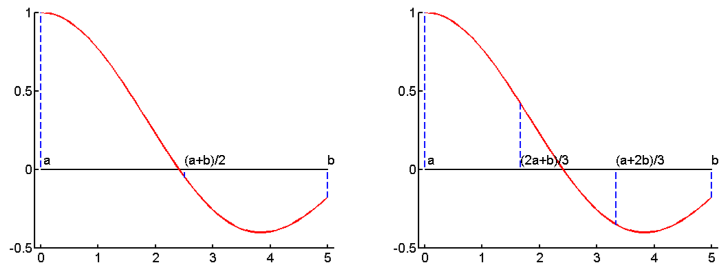
Simple bisection doesn't work:



Problem

Use the bisection strategy to compute a minimum of f.

Simple bisection doesn't work:

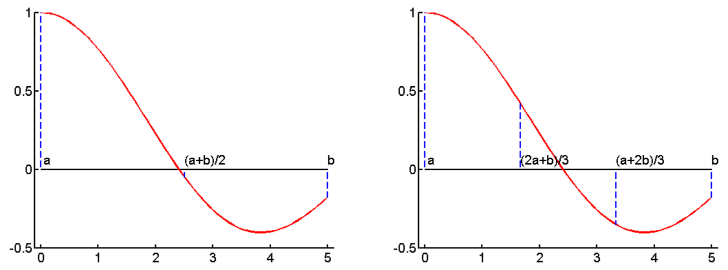


We need to trisect the interval.

Problem

Use the bisection strategy to compute a minimum of f.

Simple bisection doesn't work:



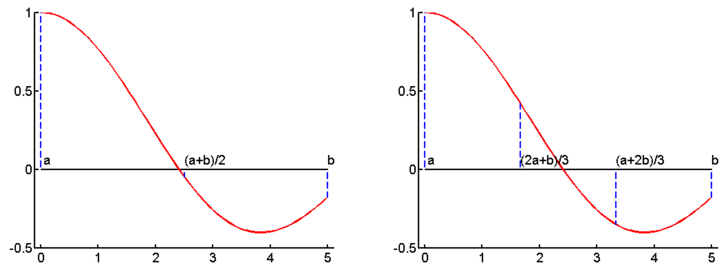
We need to trisect the interval.

Now, since $f((a+2b)/3) < f((2a+b)/3)$ we can limit our search to $[(2a+b)/3, b]$.

Problem

Use the bisection strategy to compute a minimum of f.

Simple bisection doesn't work:



We need to trisect the interval.

Now, since $f((a+2b)/3) < f((2a+b)/3)$ we can limit our search to $[(2a+b)/3, b]$.

This strategy would work, but is inefficient since $(a+2b)/3$ can't be used for the next trisection step.

Golden Section Search

Want: an efficient trisection algorithm.

Golden Section Search

Want: an efficient trisection algorithm.

What to do: pick the two interior trisection points so that they can be re-used in the next iteration (along with their associated function values, which may have been costly to obtain).

Golden Section Search

Want: an efficient trisection algorithm.

What to do: pick the two interior trisection points so that they can be re-used in the next iteration (along with their associated function values, which may have been costly to obtain).

Assume interior points are

$$\begin{aligned} u &= (1-\rho)a + \rho b = a + \rho(b-a) \\ v &= \rho a + (1-\rho)b = b - \rho(b-a), \end{aligned}$$

where $0 < \rho < 1$ is a ratio to be determined.

Golden Section Search

Want: an efficient trisection algorithm.

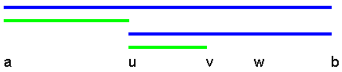
What to do: pick the two interior trisection points so that they can be re-used in the next iteration (along with their associated function values, which may have been costly to obtain).

Assume interior points are

$$\begin{aligned} u &= (1-\rho)a + \rho b = a + \rho(b-a) \\ v &= \rho a + (1-\rho)b = b - \rho(b-a), \end{aligned}$$

where $0 < \rho < 1$ is a ratio to be determined.

If, for example, the interval in the next iteration is $[u, b]$ with interior point v, then we want ρ to be such that the position of v relative to u and b is the same as that of u was to a and b in the previous iteration.



Golden Section Search (cont.)

Therefore, we want
$$\frac{b-u}{v-u} = \frac{b-a}{u-a}$$

Golden Section Search (cont.)

Therefore, we want

$$\frac{b-u}{v-u} = \frac{b-a}{u-a}$$

$$\overset{\text{Def } u,v}{\Longleftrightarrow} \quad \frac{b-(a+\rho(b-a))}{(b-\rho(b-a))-(a+\rho(b-a))} = \frac{b-a}{(a+\rho(b-a))-a}$$

Golden Section Search (cont.)

Therefore, we want
$$\begin{aligned}
\frac{b-u}{v-u} &= \frac{b-a}{u-a} \\
\overset{\text{Def } u,v}{\iff} \quad \frac{b-(a+\rho(b-a))}{(b-\rho(b-a))-(a+\rho(b-a))} &= \frac{b-a}{(a+\rho(b-a))-a} \\
\iff \quad \frac{(b-a)(1-\rho)}{(b-a)(1-2\rho)} &= \frac{b-a}{\rho(b-a)}
\end{aligned}$$

Golden Section Search (cont.)

Therefore, we want

$$
\begin{aligned}
\frac{b-u}{v-u} &= \frac{b-a}{u-a} \\
\overset{\text{Def } u,v}{\iff} \quad \frac{b-(a+\rho(b-a))}{(b-\rho(b-a))-(a+\rho(b-a))} &= \frac{b-a}{(a+\rho(b-a))-a} \\
\iff \quad \frac{(b-a)(1-\rho)}{(b-a)(1-2\rho)} &= \frac{b-a}{\rho(b-a)} \\
\iff \quad \frac{(1-\rho)}{(1-2\rho)} &= \frac{1}{\rho}
\end{aligned}
$$

Golden Section Search (cont.)

Therefore, we want

$$\begin{aligned}
\frac{b-u}{v-u} &= \frac{b-a}{u-a} \\
\overset{\text{Def } u,v}{\iff} \quad \frac{b-(a+\rho(b-a))}{(b-\rho(b-a))-(a+\rho(b-a))} &= \frac{b-a}{(a+\rho(b-a))-a} \\
\iff \quad \frac{(b-a)(1-\rho)}{(b-a)(1-2\rho)} &= \frac{b-a}{\rho(b-a)} \\
\iff \quad \frac{(1-\rho)}{(1-2\rho)} &= \frac{1}{\rho} \\
\iff \quad \rho(1-\rho) &= 1-2\rho
\end{aligned}$$

Golden Section Search (cont.)

Therefore, we want

$$
\begin{aligned}
&& \frac{b-u}{v-u} &= \frac{b-a}{u-a} \\
\overset{\text{Def } u,v}{\iff} && \frac{b-(a+\rho(b-a))}{(b-\rho(b-a))-(a+\rho(b-a))} &= \frac{b-a}{(a+\rho(b-a))-a} \\
\iff && \frac{(b-a)(1-\rho)}{(b-a)(1-2\rho)} &= \frac{b-a}{\rho(b-a)} \\
\iff && \frac{(1-\rho)}{(1-2\rho)} &= \frac{1}{\rho} \\
\iff && \rho(1-\rho) &= 1-2\rho \\
\iff && \rho^2-3\rho+1 &= 0
\end{aligned}
$$

Golden Section Search (cont.)

Therefore, we want

$$\begin{aligned}
& & \frac{b-u}{v-u} &= \frac{b-a}{u-a} \\
&\overset{\text{Def } u,v}{\iff} & \frac{b-(a+\rho(b-a))}{(b-\rho(b-a))-(a+\rho(b-a))} &= \frac{b-a}{(a+\rho(b-a))-a} \\
&\iff & \frac{(b-a)(1-\rho)}{(b-a)(1-2\rho)} &= \frac{b-a}{\rho(b-a)} \\
&\iff & \frac{(1-\rho)}{(1-2\rho)} &= \frac{1}{\rho} \\
&\iff & \rho(1-\rho) &= 1-2\rho \\
&\iff & \rho^2-3\rho+1 &= 0
\end{aligned}$$

The solution in $(0,1)$ is

$$\rho = \frac{3-\sqrt{5}}{2} \approx 0.381966.$$

Golden Section Search (cont.)

Therefore, we want

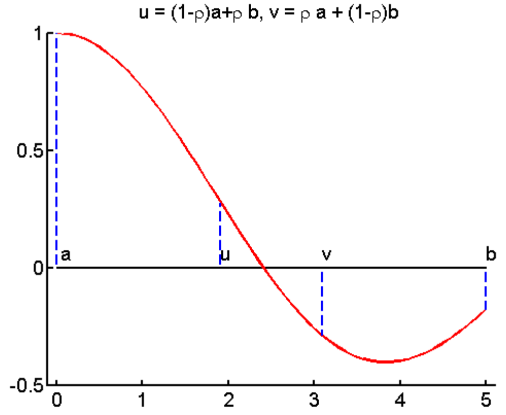
$$
\begin{aligned}
\frac{b-u}{v-u} &= \frac{b-a}{u-a} \\
\overset{\text{Def } u,v}{\iff} \quad \frac{b-(a+\rho(b-a))}{(b-\rho(b-a))-(a+\rho(b-a))} &= \frac{b-a}{(a+\rho(b-a))-a} \\
\iff \quad \frac{(b-a)(1-\rho)}{(b-a)(1-2\rho)} &= \frac{b-a}{\rho(b-a)} \\
\iff \quad \frac{(1-\rho)}{(1-2\rho)} &= \frac{1}{\rho} \\
\iff \quad \rho(1-\rho) &= 1-2\rho \\
\iff \quad \rho^2-3\rho+1 &= 0
\end{aligned}
$$

The solution in $(0,1)$ is

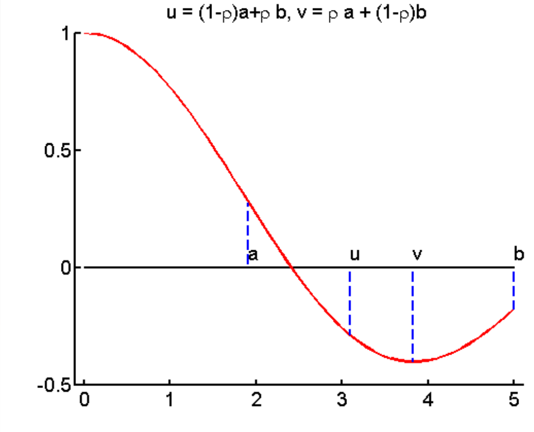
$$\rho = \tfrac{3-\sqrt{5}}{2} \approx 0.381966.$$

Since $\rho = 2-\phi$, where $\phi = \tfrac{1+\sqrt{5}}{2} \approx 1.618034$ is the golden ratio, the method is called the golden section search.

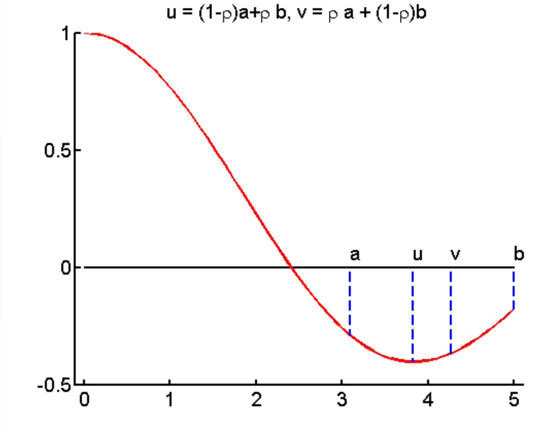
Golden Section Search (cont.)



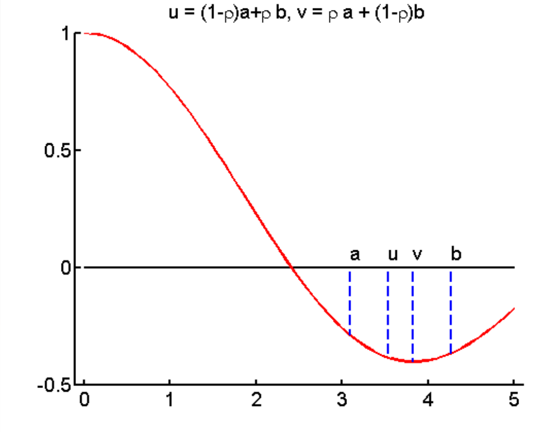
Golden Section Search (cont.)



Golden Section Search (cont.)



Golden Section Search (cont.)



While golden section search is a fool-proof algorithm that will always find the minimum of a *unimodular*[1] continuous function provided the initial interval $[a, b]$ is chosen so that it contains the minimum, it is very slow. To reduce the interval length to machine accuracy `eps`, 75 iterations are required.

---

[1] a function is unimodular if it has a single extremum on $[a, b]$

While golden section search is a fool-proof algorithm that will always find the minimum of a *unimodular*[1] continuous function provided the initial interval $[a, b]$ is chosen so that it contains the minimum, it is very slow. To reduce the interval length to machine accuracy `eps`, 75 iterations are required.

A faster — and just as robust — algorithm consists of

- golden section search (if necessary),
- parabolic interpolation (when possible).

---

[1] a function is unimodular if it has a single extremum on $[a, b]$

While golden section search is a fool-proof algorithm that will always find the minimum of a *unimodular*[1] continuous function provided the initial interval $[a, b]$ is chosen so that it contains the minimum, it is very slow. To reduce the interval length to machine accuracy `eps`, 75 iterations are required.

A faster — and just as robust — algorithm consists of

- golden section search (if necessary),
- parabolic interpolation (when possible).

This algorithm, called `fminbnd` in MATLAB, is also due to Richard Brent.

---

[1] a function is unimodular if it has a single extremum on $[a, b]$

While golden section search is a fool-proof algorithm that will always find the minimum of a *unimodular*[1] continuous function provided the initial interval $[a, b]$ is chosen so that it contains the minimum, it is very slow. To reduce the interval length to machine accuracy `eps`, 75 iterations are required.

A faster — and just as robust — algorithm consists of

- golden section search (if necessary),
- parabolic interpolation (when possible).

This algorithm, called `fminbnd` in MATLAB, is also due to Richard Brent.

If f has several minima on $[a, b]$, then `fminbnd` may not find the global minimum.

---

[1] a function is unimodular if it has a single extremum on $[a, b]$

While golden section search is a fool-proof algorithm that will always find the minimum of a *unimodular*[1] continuous function provided the initial interval $[a, b]$ is chosen so that it contains the minimum, it is very slow. To reduce the interval length to machine accuracy `eps`, 75 iterations are required.

A faster — and just as robust — algorithm consists of

- golden section search (if necessary),
- parabolic interpolation (when possible).

This algorithm, called `fminbnd` in MATLAB, is also due to Richard Brent.

If f has several minima on $[a, b]$, then `fminbnd` may not find the global minimum.

For an illustration see the MATLAB script `FminDemo.m` which calls `fmintx.m` from [NCM].

---

[1] a function is unimodular if it has a single extremum on $[a, b]$

An alternative approach

One could also use Newton's method to find the critical points of f. However, then not only f' needs to be known, but also f''.

An alternative approach

One could also use Newton's method to find the critical points of f.
However, then not only f' needs to be known, but also f''.
The iteration formula to find a critical point would be

$$x_{n+1} = x_n - \frac{f'(x_n)}{f''(x_n)}, \qquad n = 0, 1, 2, \ldots,$$

with initial guess x_0.

An alternative approach

One could also use Newton's method to find the critical points of f.
However, then not only f' needs to be known, but also f''.
The iteration formula to find a critical point would be

$$x_{n+1} = x_n - \frac{f'(x_n)}{f''(x_n)}, \qquad n = 0, 1, 2, \ldots,$$

with initial guess x_0.

Minimization of functions of more than one variable can be attempted with `fminsearch` in basic MATLAB, and with other — more powerful — functions provided in the optimization toolbox.

References I

C. Moler.
Numerical Computing with MATLAB.
SIAM, Philadelphia, 2004.
Also http://www.mathworks.com/moler/.