Introduction to Gradient-Based Optimisation

Part 2: Univariate methods

Dr. J.-D. Müller
School of Engineering and Materials Science,
Queen Mary, University of London
j.mueller@qmul.ac.uk

UK Fluids Network SIG on Numerical Optimisation with Fluids
Cambridge, 8-10 August 2018

Notes

Outline

Notes

Organisation of the lectures

1. Univariate optimisation
   - Bisection
   - Steepest Descent
   - Newton's method
2. Multivariate optimisation
   - Steepest descent and line-search methods:
   - Wolfe and Armijo conditions,
   - Newton's method, Trust-region methods,
   - Conjugate Gradient, Truncated Newton's, Quasi-Newton methods,
3. Constrained Optimisation:
   - Projected gradient methods,
   - Penalty methods,
   - Exterior and interior point methods, SQP
4. Adjoint methods
   - Reversing time
   - Automatic Differentiation
   - Adjoint CFD codes

Notes

Outline

Notes

Tank design

Properties of an open-topped tank with height x_1, sides x_2, x_3:

$$\text{Volume of a tank:} \quad V = x_1 x_2 x_3 \tag{1}$$

$$\text{Surface:} \quad S = 2x_1x_2 + 2x_1x_3 + x_2x_3 \tag{2}$$

$$\text{Minimise} \quad S \tag{3}$$

Constrained optimisation:

$$\text{Minimise} \quad S \quad \text{subject to} \quad V = V^* \tag{4}$$

We can express this constraint by eliminating one of the variables,

$$x_3 = V^* x_1^{-1} x_2^{-1}$$

Unconstrained optimisation:

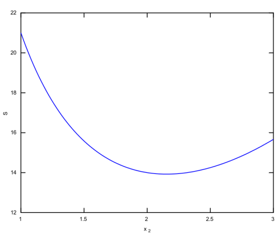
$$\text{Min} \quad S = 2x_1x_2 + 2V^*x_2^{-1} + V^*x_1^{-1}. \tag{5}$$

Notes

Univariate tank design

To simplify the problem further, assume a square base, $x_2 = x_3$. Then

$$\begin{aligned} V &= x_1 x_2^2 \\ S &= 4x_1x_2 + x_2^2 \\ \text{or with} V = V^* : \quad S &= 4V^*x_2^{-1} + x_2^2. \end{aligned}$$



Notes

Outline

Notes

Optimality conditions

Notes

From simple calculus: a local minimum exists for $F(x)$ if

$$\frac{dF}{dx} = F'(x) = 0 \quad \text{and} \quad \frac{d^2F}{dx^2} = F''(x) > 0 \tag{6}$$

If (**??**) is satisfied for $x = x^*$ and

$$F(x) \geq F(x^*) \quad \text{for all} \quad x, \tag{7}$$

then x^* is a global minimum.

Outline

Notes

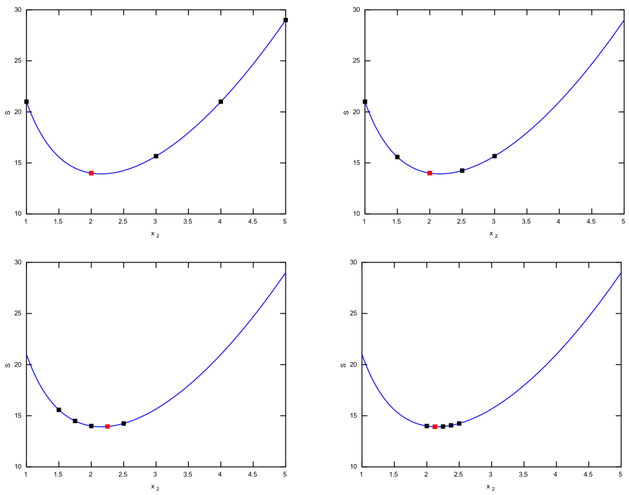
The bisection method

Simple (but inefficient) idea: given a *bracketing* interval, i.e. it contains a minimum, $a \le x^* \le b$, successively half the interval around the minimum (see Bartholomew-Biggs, section 2.2).

set $x_a = a, x_b = b$
do
  compute $F_a = F(x_a), F_b = F(x_b)$
  set $x_M = \frac{1}{2}(x_a + x_b), x_l = \frac{1}{2}(x_a + x_M), x_r = \frac{1}{2}(x_M + x_b)$
  compute $F_l = F(x_l),, F_m = F(x_m), F_r = F(x_r)$
  compute $F_{min} = \min\{F_a, F_l, F_m, F_r, F_b\}$
  if $F_{min} = F_a$ or $F_{min} = F_l$ then $x_b = x_M$,
  else if $F_{min} = F_m$ then $x_a = x_l, x_b = x_r$,
  else $x_a = x_M$.
while $|x_b - x_a| \ge \varepsilon$

Notes

Bisection on tank design problem I



Notes

Bisection on tank design problem II

$$\min S = 4V^* x_2^{-1} + x_2^2 \quad \text{with} \quad V^* = 5$$

iter	xA	xL	xM	xR	xB	fmin
1,	1.0000	2.0000	3.0000	4.0000	5.000000	14.000000
2,	1.0000	1.5000	2.0000	2.5000	3.000000	14.000000
3,	1.5000	1.7500	2.0000	2.2500	2.500000	13.951389
4,	2.0000	2.1250	2.2500	2.3750	2.500000	13.927390
5,	2.0000	2.0625	2.1250	2.1875	2.250000	13.927390
6,	2.0625	2.0938	2.1250	2.1562	2.187500	13.924776
7,	2.1250	2.1406	2.1562	2.1719	2.187500	13.924776
8,	2.1406	2.1484	2.1562	2.1641	2.171875	13.924776
9,	2.1484	2.1523	2.1562	2.1602	2.164062	13.924776
10,	2.1523	2.1543	2.1562	2.1582	2.160156	13.924767
11,	2.1523	2.1533	2.1543	2.1553	2.156250	13.924767
12,	2.1533	2.1538	2.1543	2.1548	2.155273	13.924767

Notes

Properties of the bisection method

- User has to specify the initial bracketing interval $x_A \leq x \leq x_B$ which needs to contain a minimum, although there are algorithms for this (see B-B, 2.2).
- The algorithm finds any minimum in the bracket, not necessarily the lowest minimum in the bracket
- Convergence to the optimum is rather slow and depends on the width of the initial bracket and the sought width of the final bracket ε:

$$N \geq \frac{\log_{10}(x_B - x_A) + \log_{10}\varepsilon}{\log_{10}(2)}$$

- The bisection-method is 'gradient-free', we do not need to compute gradients for it.

Notes

Outline

Notes

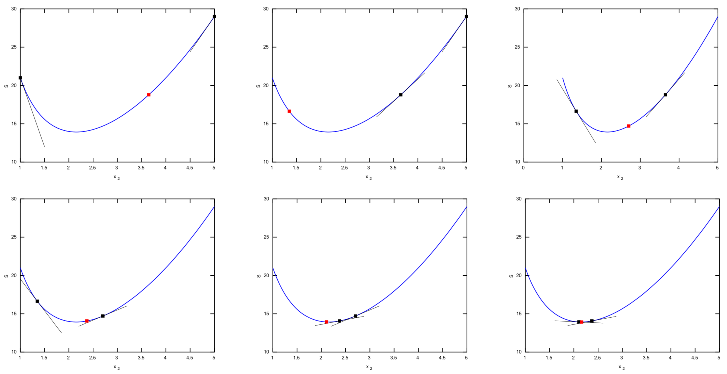
The secant method

Again, start from a bracketing interval but use gradient information to estimate the location of the minimum in the bracket.
Bracketing implies here that $x_a < x_b$, $F'(x_a) < 0$, $F'(x_b) > 0$ and $F'' > 0$. Also, assume F is twice continuously differentiable.

set $x_1 = a, x_2 = b$
    compute $F'_1 = F'(x_1), F'_2 = F'(x_2)$
set $k = 2$
do
    set $k = k + 1$
    ! *Use linear interpolation to find* $F'(x_k) = 0$ *using* x_{k-1}, x_{k-2}
    set $x_k = x_{k-2} - \frac{F'_{k-2}}{F'_{k-1} - F'_{k-2}}(x_{k-1} - x_{k-2})$
    compute $F'_k = F'(x_k)$
while $|F'_k| \geq \varepsilon$

Notes

Secant method on tank example



Notes

Secant method on tank example

iter	x_{k-2}	x_{k-1}	x	$f'(k)$	$f(k)$
3,	1.0000	5.0000	3.647059	5.790476	18.784909
4,	5.0000	3.6471	1.349327	-8.286233	16.642888
5,	3.6471	1.3493	2.701883	2.664106	14.702418
6,	1.3493	2.7019	2.372820	1.193416	14.059064
7,	2.7019	2.3728	2.105796	-0.298623	13.931973
8,	2.3728	2.1058	2.159240	0.028765	13.924836
9,	2.1058	2.1592	2.154544	0.000655	13.924767
10,	2.1592	2.1545	2.154434	-0.000001	13.924767
11,	2.1545	2.1544	2.154435	0.000000	13.924767

Notes

Alternatives for the computation of x_k

The example computed x_k from x_{k-1}, x_{k-2}, regardless of how the iterates fell around the minimum. Alternatively we could also memorise x_{k-3} once $k > 3$ and for the oldest point

- choose whichever x_{k-2}, x_{k-3} gives the smaller $|F'|$ (choose the point closer to the minimum),
- choose x_{k-2}, x_{k-3} to have the sign of F' opposite to F'_{k-1} (choose the point to bracket the minimum and interpolate rather than extrapolate).

Notes

Alternatives for the computation of x_k

iter	x_{k-2}	x_{k-1}	x	$f'(k)$	$f(k)$
chronological:					
9,	2.1058	2.1592	2.154544	0.000655	13.924767
10,	2.1592	2.1545	2.154434	-0.000001	13.924767
11,	2.1545	2.1544	2.154435	0.000000	13.924767
smallest gradient (a):					
9,	2.2177	2.1623	2.154208;	-0.001363	13.924767
10,	2.1623	2.1542	2.154436;	0.000005	13.924767
11,	2.1542	2.1544	2.154435;	0.000000	13.924767
bracketing (b):					
9,	2.2177	2.1399	2.154859;	0.002547	13.924767
10,	2.2177	2.1549	2.154422;	-0.000074	13.924767
11,	2.1549	2.1544	2.154435;	0.000000	13.924767

Notes

Properties of the secant method

- Needs computation of gradients,
- Works with first derivatives only, could converge to a maximum if the assumption that $F'' > 0$ in $[a, b]$ is violated,
- Converges better than the bisection method,
- Flexibility in how to choose x_k based on $x_{k-1}, x_{k-2}, ..,$
- Can be generalised to multi-variate problems, is the basis for some important methods such as steepest-descent and BFGS.

Notes

Higher-order secant methods

- The secant method as described performs linear interpolation on the gradient values at the end of the bracketing interval: hence it reconstructs a quadratic.
- However, with F_{k-1}, F_{k-2} and F'_{k-1}, F'_{k-2} we have 4 pieces of data, so we could reconstruct a cubic.
- Using also the new value Fk and $F'k$ we have 6 pieces of data, so we could reconstruct a quintic.
- Higher-order polynomial fits exhibit strong oscillations as the polynomial is forced to interpolate the data points, rather than approximate them. So in practice, use higher-order only if the function is found to be locally uni-modal

Notes

Outline

Notes

Newton's method

The Newton-Raphson method (actually due to Simpson in this form) finds zeroes of function by using Taylor expansion

$$0 = F(x+h) = F(x) + hF'(x) + \frac{1}{2}h^2F''(x) + O(h^3). \qquad (8)$$

Differentiating (**??**) w.r.t. h allows us to find zeroes of the gradient.

$$0 = F'(x+h) = F'(x) + hF''(x) + \frac{1}{2}h^2F'''(x) + O(h^3).$$

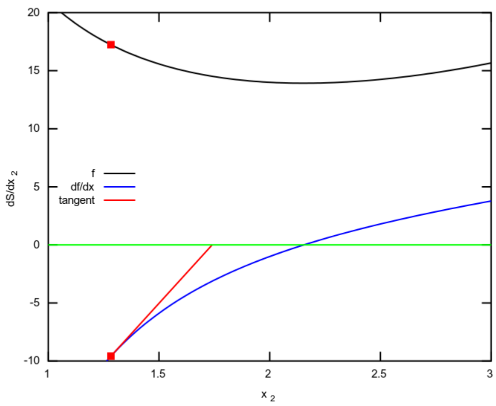
After neglecting higher terms and using $x = x_k, x_{k+1} = x_k + h$

$$x_{k+1} = x_k - \frac{F'(x)}{F''(x)}$$

Notes

Interpretation of Newton's method

Newton's method can be interpreted as using the tangent of the gradient to find the zero of the gradient.



Gradient of the tank surface S: dS/dx_2.

Notes

Convergence of Newton's method

Assuming that we are in a close neighbourhood of the minimum of a continuous and differentiable function, i.e.

- the second derivative $F'' > 0$,
- and the third derivatives are bounded by some value M

We can then show that the error of successive iterates $e_k = x^* - x_k$ are related as

$$e_{k+1} = e_k^2 \frac{F'''}{F''}$$

i.e. the error reduces quadratically with each iteration.

Newton's method has quadratic convergence

Notes

Newton's method: univariate tank example

iter	x	f''	f'	f
1,	1.0000	42.000000;	-18.000000	21.000000
2,	1.4286	15.720000;	-6.942857	16.040816
3,	1.8702	8.114707;	-1.977493	14.191634
4,	2.1139	6.234415;	-0.247768	13.929753
5,	2.1537	6.004299;	-0.004629	13.924768
6,	2.1544	6.000002;	-0.000002	13.924767
7,	2.1544	6.000000;	-0.000000	13.924767

Notes

Difficulties with Newton's method

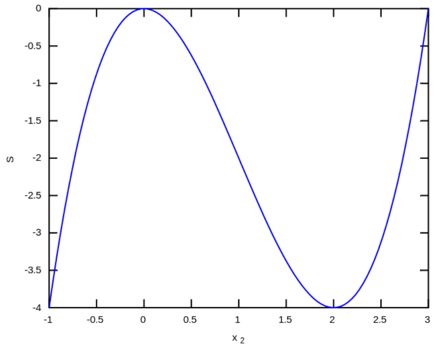
- What if $F'' < 0$? Newton's method will happily converge to a maximum. All it is concerned about is to reduce the gradient, not to maximise the second derivative.
- What if h is so large that $F''_{k+1} > 0$? Newton's method may recover in the next step, but large steps may lead outside of the validity of F.
- What if $F'' = 0$? Division by zero! Will occur for a linear univariate function or a saddlepoint in multivariate functions.

Notes

Example of unstable Newton's method

Minimise $F(x)$ varying x:

$$\min_x F(x) = x^3 - 3x^2$$



Notes

Example of unstable Newton's method

Minimise $F(x)$ varying x:

$$\min_x F(x) = x^3 - 3x^2$$

The second derivative is $F''(x) = 6x - 6$. A starting value of $x_1 = 1$ leads to division by zero:

iter	x	f''	f'	f
warning: division by zero				
1,	1.0000	0.000000;	-3.000000	-2.000000
2,	Inf	Inf;	NaN	NaN

(Inf stands for "infinity",
NaN stands for "not a number", resulting from the division by zero.)

Notes

Cubic function: alternate starting value

Starting with $x_1 = 1.1$ to the right of the inflexion point we find the minimum:

iter	x	f''	f'	f
1,	1.1000	0.600000;	-2.970000	-2.299000
2,	6.0500	30.300000;	73.507500	111.637625
3,	3.6240	15.744059;	17.656284	8.195401
4,	2.5026	9.015318;	3.772997	-3.115397
5,	2.0840	6.504261;	0.525451	-3.978216
6,	2.0033	6.019547;	0.019579	-3.999968
7,	2.0000	6.000032;	0.000032	-4.000000
8,	2.0000	6.000000;	0.000000	-4.000000

Notes

Cubic function: alterate starting value

Starting with $x_1 = 0.9$ to the left of the inflexion point we find a maximum:

iter	x	f''	f'	f
1,	0.9000	-0.600000;	-2.970000	-1.701000
2,	-4.0500	-30.300000;	73.507500	-115.637625
3,	-1.6240	-15.744059;	17.656284	-12.195401
4,	-0.5026	-9.015318;	3.772997	-0.884603
5,	-0.0840	-6.504261;	0.525451	-0.021784
6,	-0.0033	-6.019547;	0.019579	-0.000032
7,	-0.0000	-6.000032;	0.000032	-0.000000
8,	-0.0000	-6.000000;	0.000000	-0.000000

Notes

Safeguarding Newton's method

- revert to a simpler method, e.g. secant, if $F'' = 0$.
- limit the stepwidth h to ensure $F_k < F_{k-1}$
- revert to a simpler method, e.g. secant, if $F'' = 0$.

set $a < x_1 < b$, compute F_1, F_1', F_1'', set $k = 1$
while $|F_k'| \geq \varepsilon$
 if $F'' > 0$ **then**
  set $\delta x = -F_k'/F_k''$
 else
  set $\delta x = -F_k'$! note: F'' not usable, guess step length $\alpha\delta x$
 endif
 if $\delta x < 0$ **then** $\alpha = \min(1, (a - x_k)/\delta x)$
 else $\alpha = \min(1, (b - x_k)/\delta x)$
 end if
 while $F(x_k + \alpha\delta x) > F_k$
  $\alpha = \alpha/2$
 end while
 set $x_{k+1} = x_k + \alpha\delta x$, compute $F_{k+1}, F_{k+1}', F_{k+1}''$, set $k = k+1$
end while

Notes

Summary of safeguarded Newton's method

- If we have a positive second derivative $F''(x_k) > 0$ then we can use that rate of change of the derivative to estimate a new value for the control variable x_{k+1} to find $F'(x_{k+1}) = 0$.
- If not, we need to improvise:
  - The secant method, as introduced earlier, used a bracketed interval and we had gradient values at either end x_a, x_b to find x_{k+1} to find $F'(x_{k+1}) = 0$ using linear interpolation of F'.
  - With Newton's method we'd rather avoid computing the interval end gradients, so if F'' is not usable, we only have F, F', which does not allow to approximate the step length to find $f' = 0$.
  - Typically methods start with a 'unit' step, whatever the user defines that to be.
  - The method of adjusting the step (inner while loop in the algorithm) is not very good. Better methods for finding a good step will be introduced later.

Notes

Outline

Notes

Organisation of the lectures

1. Univariate optimisation
   - Bisection
   - Steepest Descent
   - Newton's method
2. Multivariate optimisation
   - Steepest descent and line-search methods:
   - Wolfe and Armijo conditions,
   - Newton's method, Trust-region methods,
   - Conjugate Gradient, Truncated Newton's, Quasi-Newton methods,
3. Constrained Optimisation:
   - Projected gradient methods,
   - Penalty methods,
   - Exterior and interior point methods, SQP
4. Adjoint methods
   - Reversing time
   - Automatic Differentiation
   - Adjoint CFD codes

Notes

Notes