ACM40290: Numerical Algorithms

Root Finding in Multiple Dimensions

Dr Barry Wardell
School of Mathematics and Statistics
University College Dublin

Systems of Equations

We now consider fixed-point iterations and Newton's method for systems of nonlinear equations

We suppose that $F : \mathbb{R}^n \to \mathbb{R}^n$, $n > 1$, and we seek a root $\alpha \in \mathbb{R}^n$ such that $F(\alpha) = 0$

In component form, this is equivalent to

$$\begin{aligned} F_1(\alpha) &= 0 \\ F_2(\alpha) &= 0 \\ &\vdots \\ F_n(\alpha) &= 0 \end{aligned}$$

Fixed-point Iteration

For a fixed-point iteration, we again seek to rewrite $F(x) = 0$ as $x = T(x)$ to obtain:

$$x_{k+1} = T(x_k)$$

The convergence proof is the same as in the scalar case if we just replace absolute value $|.|$ with norm $\|.\|$, i.e. if

$$||T(x) - T(y)|| \leq ||x - y||, \text{ then } ||x_k - \alpha|| \leq L^k ||x_0 - \alpha||$$

Hence, as before, if T is a contraction mapping it will converge to a fixed point α.

Fixed-point Iteration

Recall that we define the **Jacobian** matrix $J_T \in \mathbb{R}^{n\times n}$

$$(J_T)_{ij} = \frac{\partial T_i}{\partial x_j}, \quad i, j = 1, \cdots, n$$

If $\| J_T(x) \|_\infty < 1$, then there is some neighbourhood of α for which the fixed-point iteration converges to α.

The proof is a natural extension of the corresponding scalar (one-dimensional) result.

Fixed-point Iteration

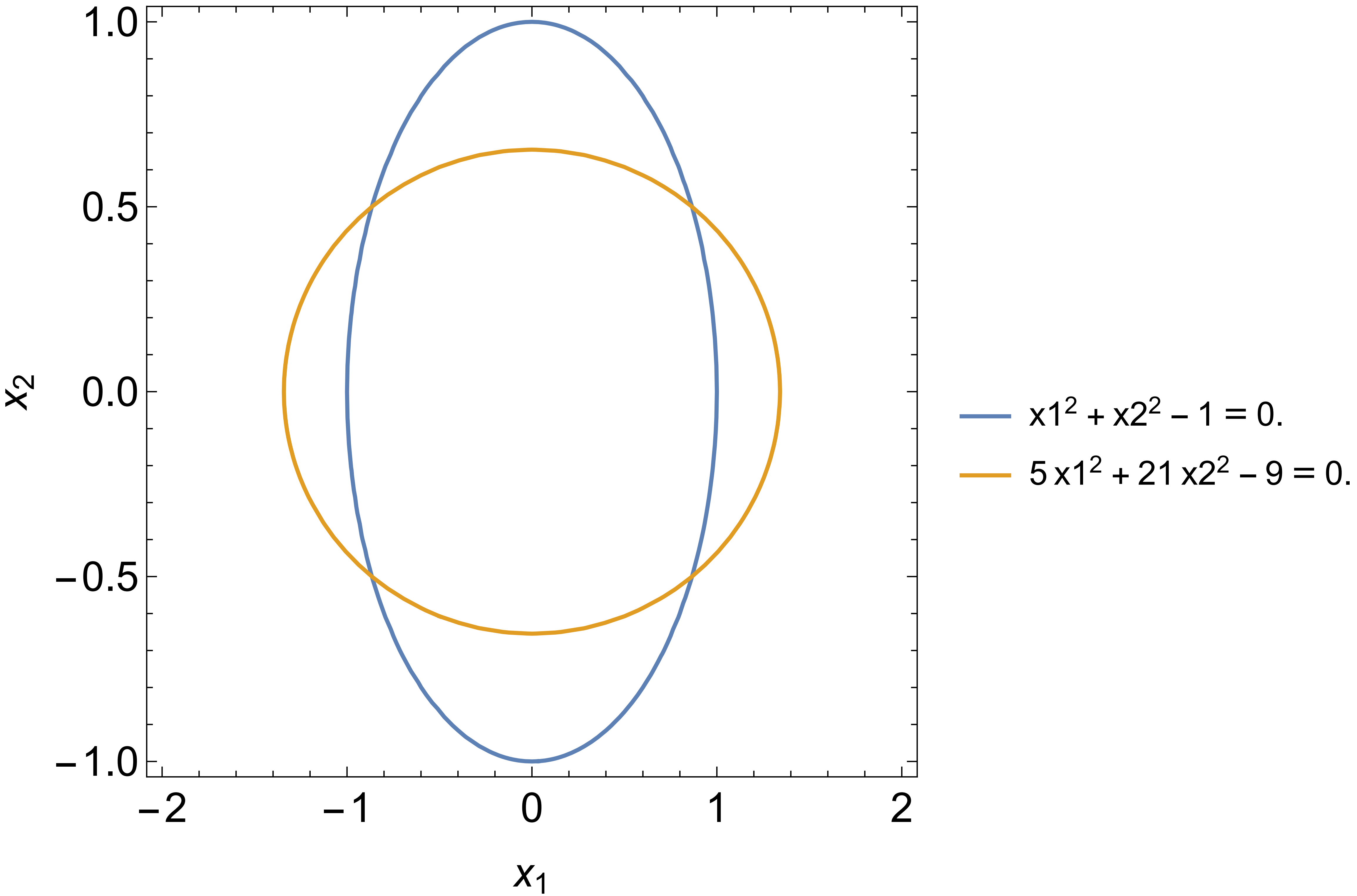
Once again, we can employ a fixed point iteration to solve $F(x) = 0$

e.g. consider

$$\begin{aligned} x_1^2 + x_2^2 - 1 &= 0 \\ 5x_1^2 + 21x_2^2 - 9 &= 0 \end{aligned}$$

This can be rearranged to $x_1 = \sqrt{1 - x_2^2}$, $x_2 = \sqrt{(9 - 5x_1^2)/21}$

Fixed-point Iteration



Fixed-point Iteration

Hence we define

$$T_1(x_1, x_2) = \sqrt{1 - x_2^2} \qquad T_2(x_1, x_2) = \sqrt{(9 - 5x_1^2)/21}$$

This yields a convergent iterative method

In this case we know the four solutions are:

$$x_1 = \pm\frac{\sqrt{3}}{2} \qquad x_2 = \pm\frac{1}{2}$$

Our iteration converges to the positive solution

Fixed-point Iteration

k	x_1	x_2
0	1	1
1	0	0.436436
2	0.899735	0.654654
3	0.755929	0.485621
4	0.87417	0.540848
5	0.84112	0.496614
...	...	...
18	0.866026	0.500002
19	0.866024	0.5
20	0.866025	0.5

Newton's Method

As in the one-dimensional case, Newton's method is generally more useful than a standard fixed-point iteration

The natural generalization of Newton's method is

$$x_{k+1} = x_k - J_F(x_k)^{-1} F(x_k), \quad k = 0, 1, 2, \ldots$$

Note that to put Newton's method in the standard form for a linear system, we write

$$J_F(x_k)\Delta x_k = -F(x_k), \quad k = 0, 1, 2, \ldots,$$

where $\Delta x_k \equiv x_{k+1} - x_k$

Want $\underline{F}(\underline{x}) = \underline{0}$

Single Equation

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

Assume $\underline{x} = [x_1 \cdots x_m]^T$

$\underline{F} = [f_1 \cdots f_m]^T$

Would like to write

$$\underline{x}^{(n+1)} = \underline{x}^{(n)} - \frac{\underline{F}(\underline{x}^{(n)})}{\underbrace{\underline{F}'(\underline{x}^{(n)})}_{\text{What do we mean?}}}$$

F' should include derivatives of f_i w.r.t x_j's (m^2 terms)

Derivatives such that

$$\underline{dF} = F'(\underline{x}^{(n)})\,\underline{\Delta x}$$

estimates change in $\underline{F}(\underline{x})$ when $\underline{x} \to \underline{x} + \underline{\Delta x}$

Scalar

$$df = \frac{\partial f}{\partial x_1}\Delta x_1 + \frac{\partial f}{\partial x_2}\Delta x_2 + \cdots \frac{\partial f}{\partial x_m}\Delta x_m$$

Suggests

$$F'(\underline{x}) = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_m} \\ \vdots & & \\ \frac{\partial f_m}{\partial x_1} & \cdots & \frac{\partial f_m}{\partial x_m} \end{bmatrix}$$

Jacobian Matrix J

<u>Then</u>

$$\underline{x}^{(n+1)} = \underline{x}^{(n)} - \left(J(\underline{x}^{(n)})\right)^{-1} \underline{F}(\underline{x}^{(n)})$$

Higher dimensional nonlinear systems

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

where $\mathbf{f} : \mathbb{R}^n \to \mathbb{R}^n$ is a given vector-valued function of n variables $x_1, \ldots x_n$.

Newton's method

$$\mathbf{x}_{n+1} = \mathbf{x}_n - \left(J(\mathbf{x}_n)\right)^{-1} \mathbf{f}(\mathbf{x}_n) \ .$$

$$J(\mathbf{x}) = \left(\frac{\partial f_i}{\partial x_j}(\mathbf{x}) \right)_{i,j=1,\ldots,n}$$

More realistically this should be written as

$$\mathbf{x}_{n+1} = \mathbf{x}_n + \mathbf{d}_n \ ,$$

the Newton correction $\mathbf{d}_n$ is computed by solving the system of n linear equations:

$$(J(\mathbf{x}_n))\mathbf{d}_n = -\mathbf{f}(\mathbf{x}_n) \ .$$

- Never compute an inverse!
- Matlab command \ can be used to solve the linear system of equations.

In practice

* Hard to construct general robust solvers in higher dimensions.
* Large problems - specialised sparse-matrix solvers need to be used.
* There are some techniques for automatic differentiation

Newton's Method Example

Example: Newton's method for the two-point Gauss quadrature rule

$$\begin{aligned}
F_1(x_1, x_2, w_1, w_2) &= w_1 + w_2 - 2 = 0 \\
F_2(x_1, x_2, w_1, w_2) &= w_1 x_1 + w_2 x_2 = 0 \\
F_3(x_1, x_2, w_1, w_2) &= w_1 x_1^2 + w_2 x_2^2 - 2/3 = 0 \\
F_4(x_1, x_2, w_1, w_2) &= w_1 x_1^3 + w_2 x_2^3 = 0
\end{aligned}$$

Newton's Method Example

We can solve this in Matlab using our own implementation of Newton's method

To do this, we require the Jacobian of this system:

$$J_F(x_1, x_2, w_1, w_2) = \begin{bmatrix} 0 & 0 & 1 & 1 \\ w_1 & w_2 & x_1 & x_2 \\ 2w_1x_1 & 2w_2x_2 & x_1^2 & x_2^2 \\ 3w_1x_1^2 & 3w_2x_2^2 & x_1^3 & x_2^3 \end{bmatrix}$$

Alternatively, we can use Matlab's built-in `fsolve` function

```
>> help fsolve
FSOLVE solves systems of nonlinear equations of
several variables.

FSOLVE attempts to solve equations of the form:

F(X) = 0 where F and X may be vectors or matrices.
```

Note that `fsolve` computes a finite difference approximation to the Jacobian by default

(Or we can pass in an analytical Jacobian if we want)

Newton's Method Example

k	x_1	x_2	w_1	w_2
0	-1	1	0.5	1.5
1	-0.333333	0.777778	1	1
2	-0.666667	0.577778	0.92	1.08
3	-0.583019	0.571873	0.985286	1.01471
4	-0.577295	0.577295	0.999996	1
5	-0.57735	0.57735	1	1