fsolve Research

1 What algorithms does `fsolve` use?

'Given a set of n nonlinear functions $F_i(x)$, where n is the number of components in the vector x, the goal of equation solving is to find a vector x that makes all $F_i(x) = 0$. `fsolve` attempts to solve a system of equations by minimizing the sum of squares of the components. If the sum of squares is zero, the system of equations is solved.'
'By default `fsolve` chooses the trust-region dogleg algorithm. The algorithm is a variant of the Powell dogleg method. The trust-region algorithm is a subspace trust-region method and is based on the interior-reflective Newton method. Each iteration involves the approximate solution of a large linear system using the method of preconditioned conjugate gradients (PCG).' (Mathworks website)

1.1 Trust Region Algorithm

Suppose that the current point is x in n-space and you want to improve by moving to a point with a lower function value. To do so, the algorithm approximates f with a simpler function q, which reasonably reflects the behavior of function f in a neighborhood N around the point x. This neighborhood is the trust region. The solver computes a trial step s by minimizing (or approximately minimizing) over N. The trust-region subproblem is

$$\min_s \{q(s), s \in N\}.$$

The solver updates the current point to $x + s$ if $f(x + s) < f(x)$; otherwise, the current point remains unchanged and the solver shrinks N (the trust region) and repeats the trial step computation.
Normal case: The quadratic approximation q is defined by the first two terms of the Taylor approximation to F at x. The neighborhood N is usually spherical or ellipsoidal in shape.
Mathematically, the trust-region subproblem is typically stated as:

$$\min \frac{1}{2} s^T H s + s^T g \text{ such that } ||Ds|| \leq \Delta,$$

where g is the gradient of f at the current point x, H is the Hessian matrix (the symmetric matrix of second derivatives), D is a diagonal scaling matrix, Δ is a positive scalar, and $|| \cdot ||$

is the 2-norm. In order to solve this equation, one can compute all eigenvalues of H and then apply a Newton process to the secular equation:

$$\frac{1}{\Delta} - \frac{1}{||s||} = 0.$$

This solves the problem accurately, however, this requires time proportional to several factorizations of H.

Therefore, an approximation has to be made. The approximation approach used here is to restricts the trust-region subproblem to a two-dimensional subspace S, in which the problem solving becomes trivial.
The solver determines the two-dimensional subspace S with the aid of a preconditioned conjugate gradient method. The solver defines S as the linear space spanned by s_1 and s_2, where s_1 is in the direction of the gradient g, and s_2 is either an **approximate Newton direction**, that is, a solution to

$$H \cdot s_s = -g,$$

or a **direction of negative curvature**,

$$s_2^T \cdot H \cdot s_2 < 0.$$

The philosophy behind this choice of S is to force global convergence (via the steepest descent direction or negative curvature direction) and achieve fast local convergence (via the Newton step, when it exists).

1.1.1 Preconditioned Conjugate Gradient Method (for getting s_2)

A popular way to solve large, symmetric, positive definite systems of linear equations $Hp = –g$ is the method of Preconditioned Conjugate Gradients (PCG). This iterative approach requires the ability to calculate matrix-vector products of the form $H \cdot v$, where v is an arbitrary vector. The symmetric positive definite matrix M is a preconditioner for H. That is, $M = C^2$, where $C^{–1}HC^{–1}$ is a well-conditioned matrix or a matrix with clustered eigenvalues.
In a minimization context, you can assume that the Hessian matrix H is symmetric. However, H is guaranteed to be positive definite only in the neighborhood of a strong minimizer. Algorithm PCG exits when it encounters a direction of negative (or zero) curvature, that is, $d^T Hd \leq 0$. The PCG output direction p is either a direction of negative curvature or an approximate solution to the Newton system $Hp = –g$. In either case, p helps to define the two-dimensional subspace used in the trust-region approach discussed in Trust-Region Methods for Nonlinear Minimization.

Optimization Book: Solving $Ax = b$ is equivalent to solving:

$$\min \phi(x) = \frac{1}{2}x^T Ax - b^T x.$$

Then

$$\nabla\phi(x) = Ax - b = r(x).$$

Conjugacy: A set of nonzero vectors $\{p_i\}$, $i = 1, ..., l$ is said to be conjugate with respect to a symmetric positive definite matrix A if

$$p_i^T Ap_j = 0,$$

for all $i \neq j$. The p_j are found iteratively and $p_0 = -r_0$. Each search direction p_k and each residual r_k is contained in the Krylov subspace of degree k for r_0:

$$\mathcal{K}(r_0, k) = span\{r_0, Ar_0, ..., A^k r_0\}.$$

1.2 Trust Region Dogleg Algorithm

Another approach is to solve a linear system of equations to find the search direction. Newton's method specifies to solve for the search direction d_k such that

$$J(x_k)d_k = -F(x_k)$$
$$x_{k+1} = x_k + d_k,$$

where $J(x_k)$ is the $n \times n$ Jacobian.
Newton's method can be problematic. $J(x_k)$ might be singular, in which case the Newton step d_k is not even defined. Also, the exact Newton step d_k can be expensive to compute. In addition, Newton's method might not converge if the starting point is far from the solution. Using trust-region techniques handles the case when $J(x_k)$ is singular and improves robustness when the starting point is far from the solution. To use a trust-region strategy, you need a merit function to decide if $x_k + 1$ is better or worse than x_k.
The Newton step d_k is a root of

$$M(x_k + d) = F(x_k) + J(x_k)d,$$

so it is also a minimum of $m(d)$, where

$$\begin{aligned} \min_d m(d) &= \frac{1}{2}||M(x_k + d)||_2^2 = \frac{1}{2}||F(x_k) + J(x_k)d||_2^2 \\ &= \frac{1}{2}F(x_k)^T F(x_k) + d^T J(x_k)^T F(x_k) + \frac{1}{2}d^T J(x_k)^T J(x_k)d. \end{aligned} \tag{1}$$

The trust region sub-problem is:

$$\min_d \left(\frac{1}{2} F(x_k)^T F(x_k) + d^T J(x_k)^T F(x_k) + \frac{1}{2} d^T J(x_k)^T J(x_k) d \right), \tag{2}$$

such that $||D \cdot d|| \leq \Delta$. You can solve this sub-problem efficiently using a dogleg strategy.

1.2.1 The dogleg implementation

The key feature of the trust-region-dogleg algorithm is the use of the Powell dogleg procedure for computing the step d, which minimizes Equation (2).
The algorithm constructs the step d from a convex combination of a **Cauchy step** (a step along the steepest descent direction) and a **Gauss-Newton step** for $f(x)$. The Cauchy step is calculated as

$$d_C = -\alpha J(x_k)^T F(x_k),$$

where α minimizes Equation (1).
The Gauss-Newton step is calculated by solving

$$J(x_k) \cdot d_{GN} = -F(x_k),$$

using the MATLAB `mldivide` (matrix left division) operator. The algorithm chooses the step d so that

$$d = d_C + \lambda(d_{GN} - d_C),$$

where λ is the largest value in the interval $[0, 1]$ such that $||d|| \leq \Delta$. If J_k is (nearly) singular, d is just the Cauchy direction.
The trust-region-dogleg algorithm is efficient because it requires only one linear solve per iteration (for the computation of the Gauss-Newton step). Additionally, the algorithm can be more robust than using the Gauss-Newton method with a line search.

1.3 Levenberg-Marquardt Method

The Levenberg-Marquardt algorithm uses a search direction that is a solution of the linear set of equations

$$(J(x_k)^T J(x_k) + \lambda_k I) d_k = -J(x_k)^T F(x_k),$$

or, optionally,

$$(J(x_k)^T J(x_k) + \lambda_k (diag J(x_k)^T J(x_k))) d_k = -J(x_k)^T F(x_k),$$

where the scalar λ_k controls both the magnitude and direction of d_k.
When λ_k is zero, the direction d_k is the Gauss-Newton method. As λ_k tends towards infinity, d_k tends towards the steepest descent direction, with magnitude tending towards zero. The implication is that, for some sufficiently large λ_k, the term $F(x_k + d_k) < F(x_k)$ holds true. Therefore, the algorithm can control the term λ_k to ensure descent despite second-order terms, which restrict the efficiency of the Gauss-Newton method. The Levenberg-Marquardt algorithm, therefore, uses a search direction that is a cross between the Gauss-Newton direction and the steepest descent direction.

Numerical Optimization Book: This method uses a trust region strategy.
Gauss-Newton Method (for $\lambda_k = 0$):

$$J_k^T J_k d_k^{GN} = -J_k^T r_k,$$

where $J_k^T J_K \approx \nabla^2 f_k$.

Lemma
The vector d is a solution of the trust-region sub-problem

$$\min_d ||Jd + r||^2, \text{ subject to: } ||d|| \leq \Delta,$$

if and only if d is feasible and there is a scalar $\lambda \geq 0$ such that

$$(J^T J + \lambda I)d = -J^T r$$
$$\lambda(\Delta - ||d||) = 0.$$

In general, this methods assumes that the objective function can be written as the sum of squares of the residuals r_j.

2 What algorithms are used in multiple shooting research?

2.1 H. G. Bock and K. J. Plitt: 'A multiple shooting algorithm for direct solution of optimal control problems'

+++ Note there is a paper that I cannot find... ('Numerical Treatment of State-constrained and Chebyshev Control Problems' (in German))+++

This paper is solving optimal control problems using **direct multiple shooting**. The algorithm presented is completely derivative free (internal numerical differentiation schemes).

The problem of interest is of the form:

$$J(x,u) = \int_{t_0}^{t_f} f(t,x(t),u(t))dt$$

subject to:

$$\begin{aligned} \dot{x} &= f(t,x(t),u(t)) \\ r(x(t_0),x(t_f)) &= 0 \text{ or } \geq 0 \\ g(t,u(t)) &\geq 0, \end{aligned}$$

where $x(t) \in \mathbf{R}^n, u(t) \in \mathbf{R}^k$. The method is able to deal with free parameters, free-end time, discontinuous right hand side, multipoint BCs (and therefore state constraints).
First time is discretized, then the control vector is approximated on each subinterval by a finite set of parameters, using basis functions ϕ_j (eg piecewise polynomials):

$$\begin{aligned} u(t) &\equiv \phi_j(t,q_j), \quad q_j \in \mathbf{R}^{k_j} \\ t &\in I_j := [t_j, t_{j+1}]. \end{aligned}$$

Then the state variable on each subinterval is replaced by its computed solution, $x(t,s_j,q_j)$, of the IVPs:

$$\begin{aligned} \dot{x} &= f(t,x,\phi_j(t,q_j)) \\ x(t_j) &= s_j, \quad t \in I_j. \end{aligned}$$

This is now a function of s_j and q_j, and so the parameter vector is $Y := (q_0,...,q_{m-1},s_0,...,s_m)$. Then the full discrete problem is:

$$\sum_{j=0}^{m-1} \int_{t_j}^{t_{j+1}} L(t,x(t,s_j,q_j),\phi_j(t,q_j))dt := J(y)$$

$$\begin{aligned} r(s_0,s_m) &= 0 \text{ or } \geq 0 \\ g_j(q_j) &\geq 0, \end{aligned}$$

Matching Condtitions:

$$x(t_{j+1},s_j,q_j) - s_{j+1} := h_j(s_j,s_{j+1},q_j) = 0.$$

2.1.1 A recursive quadratic programming method for multiple shooting

Problems like the above can be represented as:

$$\begin{aligned} F_1(y) &= \min \\ F_2(y) &= 0 \\ F_3(y) &\geq 0, \end{aligned} \tag{3}$$

which is solved by the **Quasi-Newton Method**. The idea is to improve the estimate y^k iteratively:

$$y^{k+1} = y^k + t_k \Delta y^k, \quad t_k \in [t_{min}, t_{max}].$$

The increment Δy^k is determined by the Kuhn-Tucker Solution $(\Delta y^k, \lambda_1^k, \lambda_2^k)$ of a **quadratic approximation** to (3). The quadratic approximation is:

$$\begin{aligned} \frac{1}{2}\Delta y^T B^k \Delta y + \nabla F_1^k \Delta y &:= Q_k(\Delta y) \\ F_2^k + \nabla F_2^k \Delta y &= 0 \\ F_3^k + \nabla F_3^k \Delta y &\geq 0. \end{aligned}$$

The F_i depend on y^k. The matrix B^k is defined as an approximation for the Hessian $\nabla_y^2 L$ of the Lagrange function:

$$L(y, \lambda) = F_1(y) - \lambda_1^T F_2(y) - \lambda_2^T F_3(y).$$

The matrix B is $\hat{n} \times \hat{n}$, where $\hat{n} = (m+1)n + \sum_{j=0}^{m-1} k_j$. Again, B is found iteratively:

$$B^{k+1} = B^k + U(B^k, \Delta y^k, y^k),$$

where y^k satisfies:

$$y^k = \nabla L(y^{k+1}, \lambda_1^k, \lambda_2^k) - \nabla L(y^k, \lambda_1^k, \lambda_2^k)$$

This approach works well when the Jacobian of the quadratic approximation is defined throughout and B^k is positive definite on its null-space. The method is related to generalized Gauss-Newton methods for parameter estimation in ODEs.

The quadratic approximation can be condensed by using a linearised version of the matching conditions, which permits a recursive equivalence relation on the quadratic system. The arising system, now in terms of the control parameters Δq and Δs_0, is of same size as single shooting and can be solved by a **QP Algorithm**.

It is known that the Hessian B of L has a block structure. This is of interest to be preserved, which is done by using projections, to derive a generalized update formula for B^k. The variable p^j is the projection from y space onto (s_j, q_j) or s_m-subspace. The new update formula is of rank $2(m+1)$ and structure preserving. The generalized form is:

$$B^{k+1} = B^k + \sum_{j=0}^{m} U(p^j B^K p^j, p^j, \Delta y^k, p^j \gamma^k)$$

Local super-linear convergence can be shown, when using generalized DFP-updates. This paper uses a **generalized BFGS** (a Quasi-Newton Method) update to preserve positive-definiteness.

Finally, the algorithm uses the MUSCOD Method, which uses internal numerical differentiation schemes, to avoid the calculation of any derivatives within the algorithm.

2.2 H.K. Hesse and R. Rannacher: 'Direct vs Indirect multiple shooting for nonstationary PDE-based optimization problems'

This paper is concerned with **direct and indirect multiple shooting methods**. The problems considered are of the form:

$$\begin{aligned} J(q,u) &= \min \\ &\text{such that:} \\ \partial_t u(t) + A(u(t)) + B(q(t)) &= f(t), \\ u(o) &= u_0, \end{aligned}$$

where the constraint is a parabolic PDE, A and B can be non-linear operators and u is the state and q the control variable. In particular, in weak form, we have:

$$J(q,u) = \alpha_1 J_1(u) + \alpha_2 J_2(u(T)) + \frac{1}{2}\alpha_3 ||q||_Q^2,$$

$$\text{subject to:}$$

$$((\partial_t u, \phi)) + A(u)(\phi) + B(q)(\phi) + (u(0), \phi(0)) = ((f,\phi)) + (u_0, \phi(0)),$$

where $J_1(u) = \int_I F(u)dt$, I the time domain.

2.2.1 Direct multiple shooting

The state and adjoint variables are discretized in terms of I_j, the time discretization. Furthermore, the shooting variable s^j is introduced, and the shooting nodes are denoted by τ^j. Then we have

$$((\partial_t u^j, \phi))_j + A(u^j)(\phi) + B(q^j)(\phi) + (u^j(\tau^j), \phi(\tau^j)) = ((f,\phi))_j + (s^j, \phi(\tau^j)).$$

After many calculations, the direct shooting system is:

$$\begin{aligned} (s^o - u_0, v) &= 0 \quad \forall v \in H \\ (s^{j+1} - u^j(\tau_{j+1}), v) &= 0 \quad \forall v \in H \\ (p^j - z^j(\tau_j), \psi) &= 0 \quad \forall v \in H \\ (p^m, \psi) - \alpha_2 J_2'(u^{m-1}(\tau_m))(\psi) &= 0 \quad \forall \psi \in H \\ \alpha_3 (q^j, \chi)_{Q^j} - B_q'(q^j)(\chi, z^j) &= 0 \quad \forall \chi \in Q^j, \end{aligned}$$

where p is the Lagrange multiplier, z is the dual variable.

2.2.2 Indirect multiple Shooting

The indirect multiple shooting technique derives the first order optimality system from which the following system arises:

$$\begin{aligned}(s^0 - u_0, v) &= 0 \quad \forall v \in H \\ (s^{j+1} - u^j(\tau_{j+1}), v) &= \quad \forall v \in H \\ (\lambda^j - z^j(\tau_j), v) &= \quad \forall v \in H \\ (\lambda^m, v) - \alpha_2 J_2'(s^m)(v) &= \quad \forall v \in H,\end{aligned}$$

where s^j are shooting variables as initial value for u^j and λ^{j+1} are multiple shooting variables for the final time value for z^j.

2.2.3 Numerical Methods

In both methods, **Newton's Method** is applied to the shooting system. 'This leads to a sequence of linearized systems in terms of directional derivatives of primal and dual state of the LHS and the matching conditions representing the residual on the RHS.' The paper then discusses the direct approach only and states that the indirect approach follows. Both approaches apply a **preconditioned GMRES method** for solving the linear system arising from applying Newton's Method.

In the direct approach, a condensation technique is applied to simplify the problem. This is done by reducing the system onto the space of control variables by successive substitution. These calculations result in writing the unknown Newton increments for the primal and dual shooting variables as function of the Newton increments of the control variables. Then only the control increments have to be determined. Then the solution procedure of each Newton step is brought down to finding the control increments from the derived condensed system, using GMRES. This condensation reduces computational cost by reducing the number of directional derivatives to be calculated.

Internet on GMRES Method (mathworld.wolfram.com):
An orthogonal basis $span\{r_0, Ar_0, ..., A^k r_0\}$ is formed explictly by Gram-Schmidt orthonormalization. This is applied to the Krylov sequence $\{r_0 A^k\}$, and is called Arnoldi Method. The GMRES iterates are:

$$x_i = x_0 + y_1 v_1 + ... + y_i v_i,$$

where the y_i are chosen to minimize $|b - Ax_i|$.

2.3 T. Carraro and M. Gaiger: Direct and Indirect Multiple Shooting for Parabolic Optimal Control Problems

(++ Note: also includes information from T.Carraro, M.Geiger and R. Rannacher: Indirect Multiple Shooting+++)
The paper considers OCPs of the type:

$$\min_{(q,u)} J(q,u) = \kappa_1 J_1(u) + \kappa_2 J_2(u(T)) + \frac{\alpha}{2}||q||_Q^2$$

$$\text{subject to:}$$

$$\partial_t u(x,t) + A(u(x,t)) + B(q(x,t)) = f(x,t)$$

$$u(x,0) = u_0(x).$$

The first order optimality system (KKT system) is derived using the Lagrangian method and discretized in time. Furthermore, matching conditions are derived, linking the different time intervals, via the variables s and λ. Here s^j is the initial value for u^j at τ^j and λ^{j+1} is the initial value for z^j at the point τ^{j+1}.
The KKT system is a root- finding problem and so Newton's method can be applied:

$$J_f(x_k) \cdot \delta_x = -f(x_k)$$

$$x_{k+1} = x_k + \delta_x.$$

Since f consists of derivatives of the Lagrangian, the Jacobian of f requires the computation of the Hessian of the Lagrangian. However, this system is large, and therefore **Krylov-Newton Methods** are applied to solve the system in a matrix-free manner.

2.3.1 Indirect Multiple Shooting

The idea here is to solve the interval-wise BVPs resulting from discretizing the original optimality system. In order to do this, the matching variables s and λ are fixed. There are several options how to solve the BVPs, however, this paper follows a reduced approach. There they rewrite the cost functional to be:

$$j(q^j) := J(q^j, u^j(q^j)),$$

so that J only depends on the control variable q. First, the state and adjoint equations are solved on each interval (they don't mention how...). Then the gradient of the reduced cost functional is computed. Finally, a **Newton-CG type method** (matrix free) is used to compute:

$$\nabla^2 j(q^j)\delta q^j = -\nabla j(q^j).$$

This requires the solution of two additional equations per iteration (tangent and additional adjoint equations from linearizing the forward and adjoint equation).

In a second solution step, the matching conditions in terms of s and λ are considered. This is solving a system of the form:

$$\nabla F(y_k)\delta_y = -F(y_k),$$

where $F(y) = 0$ is the shooting system where y is the set of all s^j and λ^j. This system is solved by the **Newton-GMRES type method**, which is again matrix free and again requires the solution of two additional BVPs coming from the linearizations of the state and adjoint equations with respect to s and λ.
The two solution steps are repeated iteratively until the shooting conditions are fulfilled.

2.3.2 Direct Multiple Shooting

The first solution step is to fix s^j and λ^j, as well as q^j, the control. Then, the state and adjoint variables are computed. The resulting system is not of BVP type, but of IVP type, locally because it is not fully coupled. It is possible to compute the state first and use it to compute the adjoint.
Then, in a second solution step, the matching conditions $\mathcal{L}'_{s^j}$,$\mathcal{L}'_{s^j} = 0$, $\mathcal{L}'_{\lambda^j} = 0$ and the control equation $\mathcal{L}'_{q^j} = 0$ have to be solved. This is again a system of equations that can be solved by Newton's Method. This is of the type:

$$\nabla F(q^k, s^k, \lambda^k)(\delta q^k, \delta s^k, \delta\lambda^k) = -F(q^k, s^k, \lambda^k),$$

where we are interested in solving $F(q^k, s^k, \lambda^k) = 0$. This is again done by a **Newton-GMRES type method**, which requires the renewed solution of two equations:
1. the derivatives of the state equation with respect to s^j and λ^j.
2. the derivatives of the adjoint equation with respect to all its arguments, using 1.

The direct multiple shooting approach can be changed by applying Bock's findings, reducing the problem by rewriting it in terms of s^j and q^j only (see above!).

2.4 M. Diehl, H.G. Bock and P.B. Wieber: Fast Direct Multiple Shooting Algorithms for Optimal Robot Control

This paper is about a **direct multiple shooting** approach, based on Bock et al. The paper is concerned with solving nonlinear problems of the type:

$$\min_w a(w), \text{ subject to: } b(w) = 0, \quad c(w) \geq 0,$$

where w is a finite dimensional vector representing the optimization degrees of freedom. This paper deals with ODE constraints mainly. The paper uses **iterative Sequential Quadratic Programming (SQP)/ Newton type framework**. The Lagrangian is $\mathcal{L}(w, \lambda, \mu) = a(w) - \lambda^T b(w) - \mu^T c(w)$, where λ, μ are Lagrange multipliers. The necessary condition for an optimal w^* is:

$$\begin{aligned} \nabla_w \mathcal{L}(w^*, \lambda^*, \mu^*) &= 0 \\ b(w^*) &= 0 \\ c(w^*) \geq 0, \quad \mu^* \geq 0, \quad & c(w^*)^T \mu^* = 0. \end{aligned}$$

Standard SQP interation is used:

$$\begin{aligned} w_{k+1} &= w_k + \Delta w_k, \\ \lambda_{k+1} = \lambda_k^{QP}, \quad \mu_{k+1} &= \mu_k^{QP}, \end{aligned}$$

where $(\Delta w_k, \lambda_k^{QP}, \mu_k^{QP})$ is the solution of a quadratic program. Classically, this is of the form:

$$\begin{aligned} \min_{\Delta w \in \mathbf{R}^{n_w}} \ & \frac{1}{2} \Delta w^T A_k \Delta w + \nabla_w a(w_k)^T \Delta w \\ & \text{subject to:} \\ b(w_k) + \nabla_w b(w_k)^T \Delta w &= 0 \\ c(w_k) + \nabla_w c(w_k)^T \Delta w &\geq 0, \end{aligned}$$

where $A_k \approx \nabla_w^2 \mathcal{L}$, an approximate of the Hessian of $\mathcal{L}$. Furthermore, $\nabla_w b(w_k)^T$ and $\nabla_w c(w_k)^T$ are constraint Jacobians. The Hessian can be approximated by **BFGS Updates**, or other methods. Further, different QP solvers can be used. Within each SQP iteration, the ODEs can be solved by a solver of choice.

2.5 E. Pellegrini and R.P. Russel: A Multiple-Shooting Differential Dynamic Programming Algorithm

This paper develops the MDDP (Multiple-shooting Differential Dynamic Programming) Algorithm, based on a Hybrid Differential Dynamic Programming algorithm (HDDP). The MDDP algorithm is aimed at solving multiple-phase optimal control problems and is broadly in the class of **Augmented Lagrangian Methods**. Phases are defined by a change in f dynamics,

l cost, or g constraints. The problems of interest consider **ODEs** and are of the form:

$$J = \sum_{i=1}^{m} J_i = \sum_{i=1}^{m} \int_{t_0^i}^{t_f^i} l_i(x_i, u_i, t)dt + \phi_i(x(t_0^i), t_0^i, x(t_f^i), t_f^i)$$

subject to:

$$\dot{x}^i(t) = f^i(x^i(t), u^i(t), t)$$
$$g^i(x^i(t), u^i(t), t) \leq 0$$
$$\psi(x^i(t_0^i), t_0^i, x(t_f^i), t_f^i) \leq 0,$$

where g are path constraints and ψ are boundary and inter-phase linkage conditions. Further x is the state vector, u is the control vector, f the dynamics function, l an accumulated cost, ϕ the final cost. Then each phase is split into n_l^i legs, which are the building blocks of the multiple shooting algorithm. The legs have to satisfy a simple continuity condition within each phase. The vector s gives the initial states of each of the multiple-shooting subintervals/legs.

Generally, the optimal control is found on the single leg independently of the other legs. Furthermore, it solves for the optimal initial conditions of all legs, minimizes the augmented cost and evaluates the dual functional for fixed parameters and Lagrange multipliers. The dual functional is:

$$d_c(\lambda) = \min_{u,s} \tilde{\phi}(x_n, s, \lambda, c).$$

The outer loop maximises the dual function with respect to the Lagrange multipliers. The multi leg problem involves a NLP for the s vector, using a trust region algorithm. Update initial states and targets.

Solving the single leg problem: The Lagrangian is expanded with respect to x, u and λ. The **State-Transition Matrices** (STMs) are used to obtain the necessary partials. The STMs are compute by one of three techniques: propagation of variational equations, finite differences or multi-complex step derivatives. An inverse of the shifted Hessian is computed using a trust region algorithm. Solve the quadratic subproblem using a **trust-region algorithm**. The control law is computed and the trust-region radius is updated.

2.6 H.Diedam and S.Sager: Global Optimal Control With the Direct Multiple Shooting Method

Based on Bock. Uses convex relaxations.

3 Alternatives to `fsolve`

3.1 Carraro, Geiger, Rannacher

According to their paper on indirect multiple shooting, they are solving the BVPs on each interval using **Newton CG type methods**. We are using `ode15s`. The matching conditions in this paper are solved by a **Newton-GMRES type method**. This is done by rewriting the conditions as $F(y) = 0$, where y comprises all shooting variables. In each iteration, the system:

$$\nabla F(y_k)\delta y = -F(y_k)$$

is solved. However, a preconditioner is used, since the condition of the Jacobian deteriorates with the increasing number of shooting intervals. The paper recommends a Gauss-Seidel block preconditioner (based on a paper by Heinkenschloss). There exists an inbuilt GMRES solver in Matlab.

3.2 Other Matlab Optimization Algorithms

3.2.1 `fmincon` Algorithms

`fmincon` has five inbuilt algorithms: The interior-point algorithm is the default algorithm. The other four algorithms are 'trust-region-reflective', 'sqp', 'sqp-legacy' and 'active-set'. However, `fmincon` deals with constrained nonlinear problems.

3.2.2 `fminunc` Algorithms

This solver approaches problems of the type:

$$\min_x f(x).$$

`fminunc` has two algorithms: 'quasi-newton' and 'trust-region'. For the discussion on the trust region method, see above. `fminunc` is dealing with unconstrained nonlinear problems.

The 'quasi-newton' algorithm:
Consider a quadratic model problem of the form

$$\min_x \frac{1}{2}x^T Hx + c^T x + b,$$

where the Hessian matrix, H, is symmetric positive definite, c^T a constant vector and b a constant. The optimal solution to this is when the partial derivatives of x go to zero:

$$\nabla f(x^*) = Hx^* + c = 0.$$

The optimal solution point can be written as:

$$x^* = -H^{-1}c.$$

Quasi-Newton methods avoid calculating H directly, but approximate this by observed behaviour of $f(x)$ and $\nabla f(x)$. The update of the Hessian at each iteration is given by **BFGS**:

$$H_{k+1} = H_k + \frac{q_k q_k^T}{q_k^T s_k} - \frac{H_k s_k s_k^T H_k^T}{s_k^T H_k s_k}, \tag{4}$$

where

$$s_k = x_{k+1} - x_k$$
$$q_k = \nabla f(x_{k+1}) - \nabla f(x_k).$$

'As a starting point, H_0 can be set to any symmetric positive definite matrix, for example, the identity matrix. To avoid the inversion of the Hessian H, you can derive an updating method that avoids the direct inversion of H by using a formula that makes an approximation of the inverse Hessian H^{-1} at each update. A well-known procedure is the DFP formula of Davidon, Fletcher, and Powell. This uses the same formula as the BFGS method (4) except that q_k is substituted for s_k.

The gradient information is either supplied through analytically calculated gradients, or derived by partial derivatives using a numerical differentiation method via finite differences. This involves perturbing each of the design variables, x, in turn and calculating the rate of change in the objective function.'

A line search is done at each iteration, in direction d:

$$d = -H_k^{-1}\nabla f(x_k).$$

Line search methods find the iterate x_{k+1} by computing:

$$x_{k+1} = x_k + \alpha d_k,$$

where α is the step length. The idea is to decrease the objective function along the line above, by repeatedly minimizing polynomial interpolation models of the objective function. The line search procedure has two main steps:

1. bracketing: determining the range of points on the line to be searched; an interval of α values.

2. sectioning: dividing the bracket into subintervals. Finding the minimum of the objective function (approximated by polynomial interpolation).

The resulting α satisfies the Wolfe conditions:

$$f(x_k + \alpha d_k) \leq f(x_k) + c_1 \alpha \nabla f_k^T d_k$$
$$\nabla f(x_k + \alpha d_k)^T d_k \geq c_2 \nabla f_k^T d_k,$$

where c_1, c_2 are constants in $(0, 1)$. The first condition requires that α_k sufficiently decreases the objective function. The second condition ensures that the step length is not too small.

Hessian Updates:
Many of the optimization functions determine the direction of search by updating the Hessian matrix at each iteration, using the BFGS method.The function `fminunc` also provides an option to use the DFP method. The Hessian is always positive definite, because we can initialize this and then require $q_k^T s_k$ positive. This term can always be positive, if the line search is sufficiently accurate, since it satisfies:

$$q_k^T s_k = \alpha_k(\nabla f(_{x+1})^T d - \nabla f(x_k)^T d).$$

if H is posititve definite, then d is always a descent direction (and vice versa).

3.2.3 `lsqlin` Algorithms

`lsqlin` has two algorithms: 'interior-point' and 'trust-region-reflective'. There are also some more least squared functions in Matalb, but they use the algorithms that are covered above.

3.2.4 Linear and Quadratic Programming Algorithms

Not applicable here?!

'References'

+++ Very Incomplete +++ https://uk.mathworks.com/help/optim/ug/equation-solving-algorithms.html#bro1ao1-2
https://uk.mathworks.com/help/optim/ug/fsolve.html#butbmfz-5