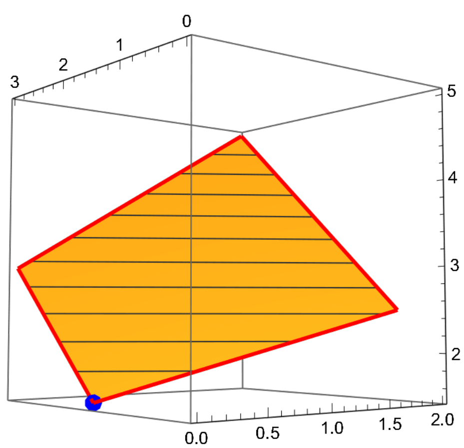
WOLFRAM U

Scientific Computing with Mathematica

PART (4): NUMERICAL OPTIMIZATION



The Basic Optimization Problem

- The basic optimization problem is

$$\textbf{minimize } f(x)$$
$$\textbf{subject to } \Phi(x)$$

- x is a design point which can be represented as a vector of values.
- The elements of x are adjusted to minimize the objective function $f(x) \in R$.
- But you have to make sure x still satisfies the constraints $\phi(x)$.

Outline

Classes of Optimization Problems

Unconstrained Optimization

Constrained Optimization

Linear Programming

Quadratic Programming

Convex Optimization

Local vs Global

- In Mathematica®, local optimization problems constrained or unconstrained can be solved using **FindMinimum**.
- Global optimization problems can be solved exactly using **Minimize** or numerically using **NMinimize**.
- In convex optimization (linear optimization is a subclass), any local solution is a global minimizer.
- Finding global solutions to large-scale optimization problems is hard (except for convex optimization).

Example

In[ ]:=
```
f[x_]:= 3x^4-28x^3+84x^2-96x+42
Plot[f[x],{x,0,5}];
```

This finds the global solution symbolically:

```
Minimize[f[x],x];
```

This finds the global solution numerically:

In[ ]:=
```
NMinimize[f[x],{x,0,5}];
```

This tries to find a local solution around a pre-specified guess point:

In[ ]:=
```
guess = 3;
FindMinimum[f[x],{x,guess}];
```

Constrained vs Unconstrained

- Constrained optimization problems are problems for which a function $f(x)$ is to be minimized or maximized subject to constraints $\phi(x)$.

$$\begin{aligned}&\textbf{minimize } f(x)\\&\textbf{subject to } \Phi(x)\end{aligned}$$

- For unconstrained problems, the feasible set is R^N.
- In Mathematica, local optimization problems constrained or unconstrained can be solved using FindMinimum.

Example

This is a simple constrained optimization problem:

In[ ]:=

```
f[x_]:= 3x^4-28x^3+84x^2-96x+42
guess = 1.4;
FindMinimum[{f[x],x<1.5},{x,guess}];
```

Linear vs Nonlinear

- Linear optimization problems are optimization problems where the objective function and constraints are all linear.
- The **LinearProgramming** function in Mathematica gives direct access to linear programming algorithms and is efficient for large-scale problems.

Example

Here is a linear optimization problem.

Minimize $x + y$ subject to the constraints $x + 2y \geq 3$, $x \geq -1$, $y \geq -1$:

In[ ]:=
```
res=Minimize[x+y,{x+2y≥3,x≥-1,y≥-1},{x,y}];
```

Here is a nonlinear optimization problem with the same constrains, but with a nonlinear objective function:

In[ ]:=
```
res=Minimize[Cos[x]+y^2,{x+2y≥3,x≥-1,y≥-1},{x,y}];
```

Convex vs Non-convex

- Convex optimization problems are optimization problems where the objective function and constraints are all convex.
- There are fast and robust optimization algorithms, both in theory and in practice.
- Mathematica 12 expands to include optimization of convex functions over convex constraints.
- The **ConvexOptimization** function in Mathematica gives direct access to linear programming algorithms and is efficient for large-scale problems.

Example

Minimize $|x + 2y|$ subject to linear constraints:

In[ ]:=
```
ConvexOptimization[Abs[x + 2 y], {x + y == 1., x - y ≤ 2}, {x, y}];
```

Classification of Numerical Optimization Algorithms

Gradient-based methods

Gradient search methods use first derivatives (gradients) or second derivatives (Hessians) information.

- The sequential linear programming method
- The sequential quadratic programming method
- Penalty methods
- The augmented Lagrangian method
- The (nonlinear) interior point method

Direct search methods

Direct search methods do not use derivative information.

- Genetic algorithm and differential evolution
- Simulated annealing

Unconstrained Optimization

The essence of most methods is in the local quadratic model

$$q_k(p) = f(x_k) + \nabla f(x_k)^T p + \frac{1}{2} p^T B_k p$$

which uses the Hessian information to approximate the right step to reach the local minimum.

The FindMinimum function has five different methods:

In[ ]:=
```
TextGrid[{{"Newton", "use the exact Hessian or a finite difference approximation"},
  {"QuasiNewton", "use the quasi-Newton BFGS approximation"},
  {"LevenbergMarquardt", "a Gauss-Newton method for least-squares problems"},
  {"ConjugateGradient", "a nonlinear version of the
     conjugate gradient method for solving linear systems"},
  {"PrincipalAxis", "works without using any derivatives"}}, Frame → All]
```

Out[ ]=

Newton	use the exact Hessian or a finite difference approximation
QuasiNewton	use the quasi–Newton BFGS approximation
LevenbergMarquardt	a Gauss–Newton method for least–squares problems
ConjugateGradient	a nonlinear version of the conjugate gradient method for solving linear systems
PrincipalAxis	works without using any derivatives

- With **`Method -> Automatic`**, the quasi-Newton method is used.
- If the problem is structurally a sum of squares, the Levenberg-Marquardt method is used.
- When given two starting conditions in each variable, the PrincipalAxis method is used.

Unconstrained Optimization

Newton’s Method

Local model

$$q_k(x) = f(x_k) + \nabla f(x_k)^T (x - x_k) + \frac{1}{2} (x - x_k)^T B_k (x - x_k)$$

Evaluating the gradient and setting it to zero:

$$\nabla q_k(x) = \nabla f(x_k) + B_k (x - x_k) = 0$$

Then solve for the next iterate:

$$x_{k+1} = x_k - B_k^{-1} \nabla f(x_k) B_k = 0$$

Unconstrained Optimization

Newton's Method

- With **Method → "Newton"**, the symbolic derivative will be computed automatically.
- If the function cannot be explicitly differentiated, FindMinimum uses finite differences to compute the Hessian.

This loads a package that contains some utility functions:

In[ ]:=
```
Needs["Optimization`UnconstrainedProblems`"]
```

FindMinimum will compute the Hessian symbolically:

In[ ]:=
```
FindMinimumPlot[Cos[x^2 - 3 y] + Sin[x^2 + y^2],{{x,1},{y,1}}, Method->"Newton"];
```

Unconstrained Optimization

Newton’s Method

Robustness

- The local quadratic model has to be positive definite, otherwise the method may not converge.
- For robustness, the Newton’s step is modified by a line search or by using trust region methods.

This uses the unconstrained problems package to set up the classic Rosenbrock function, which has a narrow curved valley:

```
p = GetFindMinimumProblem [Rosenbrock ];
```

In[ ]:=
```
FindMinimumPlot [p, Method →{"Newton ","StepControl "->"LineSearch "}];
```

In[ ]:=
```
FindMinimumPlot [p, Method →{"Newton ","StepControl "->"TrustRegion "}];
```

Unconstrained Optimization

Quasi-Newton Methods

- The exact Hessian is very expensive to compute.
- The idea is to build the Hessian matrix from the function and gradient values from some steps previously taken.
- Quasi-Newton methods are chosen as the default in Mathematica.

In[ ]:=
```
FindMinimumPlot [Cos[x^2 - 3 y] + Sin[x^2 + y^2],{{x,1},{y,1}}, Method ->"QuasiNewton "];
```

For large-scale sparse problems (e.g. **Length[x] > 250**), the LBFGS (low-memory BFGS) method is used instead:

In[ ]:=
```
m = 5;
FindMinimumPlot [Cos[x^2 - 3 y] + Sin[x^2 + y^2],{{x,1},{y,1}}, Method →{"QuasiNewton ","StepMemory
```

Unconstrained Optimization

The Nonlinear Conjugate Gradient Method

- The nonlinear conjugate gradient (CG) method generalizes the conjugate gradient method to nonlinear optimization.
- CG methods converge much more slowly than “Newton” or “quasi-Newton” methods, but have lower memory requirements.

In[ ]:=
```
FindMinimumPlot[Cos[x^2 - 3 y] + Sin[x^2 + y^2],{{x,1},{y,1}}, Method→{"ConjugateGradient"}];
```

Constrained Optimization

- For general nonconvex constrained optimization problems, the interior point algorithm is used.

In[ ]:=
```
FindMinimum[{x^2+y^2 , x^3+y-x ==1 }, {{x,0}, {y,1}}, Method -> "InteriorPoint "]
```

Out[ ]=
```
{0.59985 , {x → -0.38088 , y → 0.674374}}
```

Linear Optimization

Linear optimization finds x that solves the problem:

minimize	c.x
subject to constraints	$a.x+b\geq 0$, $a_{eq}.x+b_{eq}=0$
where	$c\in\mathbb{R}^n$, $a\in\mathbb{R}^{m\times n}$, $b\in\mathbb{R}^m$, $a_{eq}\in\mathbb{R}^{k\times n}$, $b_{eq}\in\mathbb{R}^k$

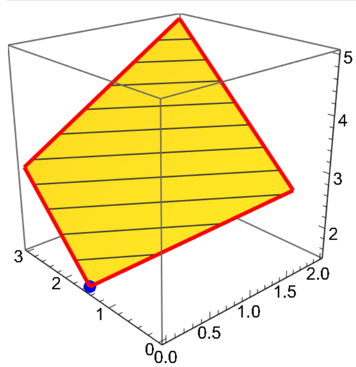
Minimize $x+y$ subject to the constraints $x+2y\geq 3$, $x\geq 0$, $y\geq 0$:

In[ ]:=
```
res=LinearOptimization [x+y,{x+2y≥3,x≥0,y≥0},{x,y}]
```

Out[ ]= $\left\{x\to 0,\ y\to \frac{3}{2}\right\}$

The optimal point lies in a region defined by the constraints and where $x+y$ is smallest within the region:

In[ ]:=
```
Show[Plot3D[x+y,{x,0,2},{y,0,3}, ⋯ ],Graphics3D [{Blue , PointSize [0.04],Point[{x,y,x+y}/.res]}]]
```

Out[ ]=


Quadratic Optimization

Quadratic optimization finds x that solves the problem:

minimize	$\frac{1}{2}\, x.q.x + c.x$
subject to constraints	$a.x + b \geq 0$, $a_{eq}.x + b_{eq} = 0$
where	$q \in \mathcal{S}_{+}^{n},\ c \in \mathbb{R}^{n},\ a \in \mathbb{R}^{m\times n},\ b \in \mathbb{R}^{m},\ a_{eq} \in \mathbb{R}^{k\times n},\ b_{eq} \in \mathbb{R}^{k}$

This minimizes $2x^2 + 20y^2 + 6xy + 5x$ subject to the constraint $-x + y \geq 2$:

In[ ]:=
```
obj=2x^2+20y^2+6x y+5x;
res=QuadraticOptimization[obj,-x+y≥2,{x,y}]
```

Out[ ]= {x → -1.73214 , y → 0.267857}

The optimal point lies in a region defined by the constraints and where $2x^2 + 20y^2 + 6xy + 5x$ is smallest:

In[ ]:=
```
Show[Plot3D[obj,{x,-2,0},{y,0,1}, ... ],Graphics3D[{Blue, PointSize[0.05],Point[{x,y,obj}/.res]}]]
```

Convex Optimization

Special Classes of Convex Optimization Problems

LinearOptimization—minimize $\{c.x \mid a.x + b \succeq 0\}$

LinearFractionalOptimization—minimize $\{(\alpha.x + \beta)/(\gamma.x + \delta) \mid a.x + b \succeq 0\}$

QuadraticOptimization—minimize $\left\{\frac{1}{2} x.q.x + c.x \,\middle|\, a.x + b \succeq 0\right\}$

SecondOrderConeOptimization—minimize $\left\{c.x \,\middle|\, \left\|a_j.x + b_j\right\| \leq \alpha_j + \beta_j,\ j = 1 \ldots k\right\}$

SemidefiniteOptimization—minimize $\left\{c.x \,\middle|\, a_0 + x_1 a_1 + \ldots + x_k a_k \succeq_{S_+^n} 0\right\}$

GeometricOptimization—minimize $\left\{\sum_{j=1}^{k_0} c_{j0} \prod_{l=1}^{n} x_l^{\alpha_{jl0}} \,\middle|\, \sum_{j=1}^{k_i} c_{ji} \prod_{l=1}^{n} x_l^{\alpha_{jli}} \leq 1,\ i = 1, \ldots, s\right\}$

ConicOptimization—minimize $\left\{c.x \,\middle|\, a_j.x + b_j \succeq_{\kappa_j} 0,\ j = 1 \ldots k\right\}$

When FindMinimum or NMinimize detects the problem as convex, they call on one of these solvers.

Convex Optimization

Special Classes of Convex Optimization Problems

The option **`Method -> method`** may be used to specify the method to use. Available methods include:

`Automatic`	`choose the method automatically`
solver	`transform the problem, if possible, to use` *solver* `to sol problem`
`"SCS"`	`SCS (splitting conic solver) library`
`"CSDP"`	`CSDP (COIN semidefinite programming) library`
`"DSDP"`	`DSDP (semidefinite programming) library`

`Method -> solver` may be used to specify that a particular solver is used.

Possible solvers are LinearOptimization, LinearFractionalOptimization, QuadraticOptimization, SecondOrderConeOptimization, SemidefiniteOptimization, ConicOptimization and GeometricOptimization.

Convex Optimization

Modelling Tools for Optimization

For vector inequality constraints:

VectorGreaterEqual—partial ordering for vectors and matrices

VectorLessEqual, **VectorGreater**, **VectorLess**

Examples

$x \succeq y$ yields **True** when $x_i \geq y_i$ is True for all $i = 1, \ldots, n$:

In[ ]:=
```
{1,2,3}≽{1,1,2}
```

Out[ ]= True

Minimize $2\,x_1 + 3\,x_2$ subject to $a_0 + a_1\,x_1 + a_2\,x_2 \succeq_{\mathcal{S}_+^2} 0$, $x_1 \in \mathbb{R}$, $x_2 \in \mathbb{R}$:

In[ ]:=
$$a_0=\begin{pmatrix}0 & 1\\1 & 0\end{pmatrix};a_1=\begin{pmatrix}1 & 0\\0 & 0\end{pmatrix};a_2=\begin{pmatrix}0 & 0\\0 & 1\end{pmatrix};$$

$$\text{SemidefiniteOptimization}\left[2x_1+3x_2,a_0+a_1x_1+a_2\,x_2\succeq_{\mathcal{S}_+^2}0,\{x_1\in\text{Reals},x_2\in\text{Reals}\}\right]$$

Out[ ]= $\{x_1 \to 1.22474,\ x_2 \to 0.816497\}$