ECE 4960: Computational and Software Engineering

Spring 2017

Note 5: Nonlinear Equations and Optimization

Reading Assignments:

1. Chaps. 6, 7, and 9, D. Bindel and J. Goodman, *Principles of Scientific Computing*, 2009.
2. Chap. 13, S. Oliveira and D. Stewart, *Writing Scientific Software: A Guide to Good Style*, Cambridge 2006.

1. Class logistics

- Programming Assisgnment 3
- Reading review 4 (will be multiple choice questions on Blackboard)

2. Equivalency between nonlinear solutions and optimization

2.1 Finding roots in multi-variate nonlinear equations and smooth optimization

Finding the roots of a set of nonlinear functions can be expressed as:

$$\begin{aligned} &f_1(x_1, x_2, ..., x_n) = 0 \\ &f_2(x_1, x_2, ..., x_n) = 0 \\ &... \\ &f_n(x_1, x_2, ..., x_n) = 0 \end{aligned} \quad \text{or in vector from:} \quad \vec{f}(\vec{x}) = 0 \qquad (1)$$

Notice that we have the same number of variables and equations so that the problem is not over or under specified. Or equivalently, both $\vec{f}$ and $\vec{x}$ are vectors of rank *n*. We define the **Jacobian matrix** *J* also of the same rank *n* to be the variation in each functional value f_j with each variable x_i:

$$J = \nabla \cdot \vec{f}(\vec{x}) = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_2}{\partial x_1} & ... & \frac{\partial f_n}{\partial x_1} \\ \frac{\partial f_1}{\partial x_2} & \frac{\partial f_2}{\partial x_2} & ... & \frac{\partial f_n}{\partial x_2} \\ ... & ... & ... & ... \\ \frac{\partial f_1}{\partial x_n} & \frac{\partial f_2}{\partial x_2} & ... & \frac{\partial f_n}{\partial x_n} \end{bmatrix} \text{ or } \quad J_{ij} = \frac{\partial f_j}{\partial x_i} \qquad (2)$$

Surely if choose to search the space $(x_1, x_2, ..., x_n)$ by the bisection method, we never have to invoke the information of the Jacobian. In 1D, bisection is to chop the search region in half to determine which

section has the root, and do it recursively. Assume that we need to do N steps to find the solution. In a space of rank n, unfortunately the search steps become N^n, which is often prohibitively large. The bisection method (in $n = 2$, it will become quad tree, and in $n = 3$, oct-tree) is most often limited to very small number of degrees of freedom (i.e., very small n).

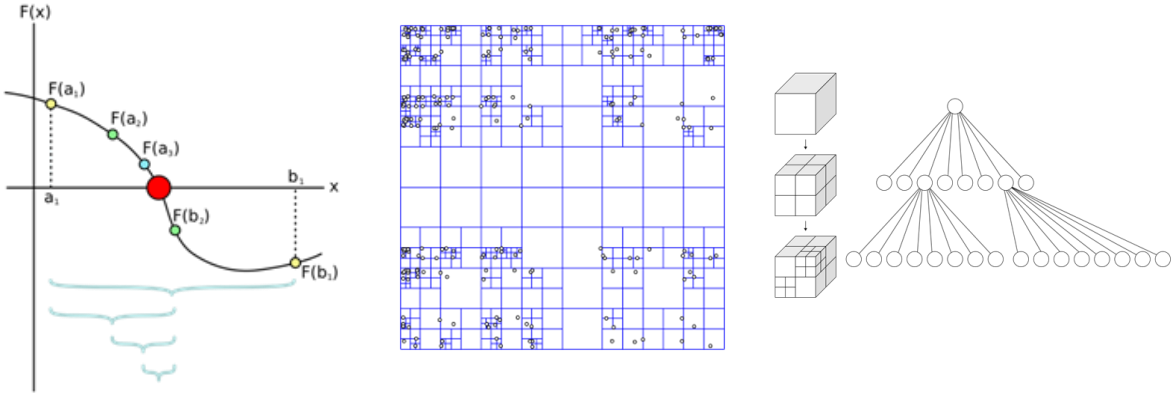


Fig. 1. Bisection, quadtree and octtree methods for searching the solution space.

Hacker Practice 5.1:

Use the bisection method to solve the following nonlinear equation:

$$f(x) = e^x - 1 = 0$$

For the initial search $x \in [-10, 10]$

For a bit more challenges, use the quad tree to solve (without variable substitution):

$$f_1(x, y) = e^x - e^y = 0;$$
$$f_2(x, y) = e^x + e^y = 2$$

For the initial search $x \in [-10, 10]$; $y \in [-10, 10]$

2.2 Optimization of Multi-Variate Nonlinear Systems

Finding the optimization of a scalar objective function $V(x_1, x_2, ..., x_n)$ can be defined by the local Taylor expansion with respect to the vector $\vec{x}$ as:

$$V(\vec{x} + \Delta\vec{x}) = V(\vec{x}) + \nabla V(\vec{x}) \cdot \Delta\vec{x} + \frac{1}{2}(\Delta\vec{x})^t \cdot [H]\Delta\vec{x} \qquad (3)$$

where the **gradient function** $\nabla V(\vec{x})$ and the **Hessian matrix** *[H]* are defined by:

$$\nabla V(x_1, x_2, ..., x_n) = \begin{pmatrix} \frac{\partial V(x_1, x_2, ..., x_n)}{\partial x_1} \\ \frac{\partial V(x_1, x_2, ..., x_n)}{\partial x_2} \\ ... \\ \frac{\partial V(x_1, x_2, ..., x_n)}{\partial x_n} \end{pmatrix} \text{ or in vector form: } \nabla V(\vec{x}) = \left(\frac{\partial V(x_1, x_2, ..., x_n)}{\partial x_j} \right) \qquad (4)$$

$$[H] = \nabla \cdot \nabla V(\vec{x}) \qquad \text{or} \qquad H_{ij} = \frac{\partial \left(\frac{\partial V(x_1, x_2, ..., x_n)}{\partial x_j} \right)}{\partial x_i} \qquad (5)$$

Minimization of the scalar objective function $V(x_1, x_2, ..., x_n)$ is then defined as $\forall \|\Delta \vec{x}\| < R$, we have $V(\vec{x} + \Delta \vec{x}) > V(\vec{x})$ when $\nabla V(\vec{x}) = 0$ (as we cannot guarantee the sign or direction of $\Delta \vec{x}$) and $(\Delta \vec{x})^t \cdot [H] \Delta \vec{x} > 0$, where *R* is the range within which $V(x_1, x_2, ..., x_n)$ has a local minimum.

$V(x_1, x_2, ..., x_n)$ has a global minimum when $R \to \infty$. Either minimization or maximization will depend on the sign of the last term in Eq. (3) $(\Delta \vec{x})^t \cdot [H] \Delta \vec{x}$ to be positive or negative. When *[H]* is positive definite, $V(x_1, x_2, ..., x_n)$ has a local minimum, when *[H]* is negative definite, $V(x_1, x_2, ..., x_n)$ has a local maximum.

Notice that how the gradient and dot product are defined similarly to the vector space, which makes every term in Eq. (3) to be scalar.

Multiple objective functions are often put together by the **Langrangian multipliers**:

$$V(x_1, x_2, ..., x_n) = \sum_i \lambda_i V_i(x_1, x_2, ..., x_n) \qquad (6)$$

Another possibility of multiple objective functions will use the norm as: $\|(V_1, V_2, \ldots V_m)\|$. If the second norm is used, this is often referred as the "least square".

The **equivalence** of the two problems can be viewed by optimization of the scalar objective function $V(x_1, x_2, ..., x_n)$ to be finding the roots of $\nabla V(\vec{x}) = 0$, and the root finding of the nonlinear equation $\vec{f}(\vec{x}) = 0$ can be viewed as minimization of $\left\| \vec{f}(\vec{x}) \right\|_2$. For sure, all of the derivatives defined require the function to be smooth, which we will mention the exception explicitly if we have to deal with discontinuities.

When $\vec{f}(\vec{x})$ and $V(x_1, x_2, ..., x_n)$ cannot be expressed in analytical forms but can be evaluated with given $(x_1, x_2, ..., x_n)$, or otherwise we have no control of $\vec{f}(\vec{x})$ or $V(x_1, x_2, ..., x_n)$, the root finding and the optimization problems have the "black boxes" of $\vec{f}(\vec{x})$ and $V(x_1, x_2, ..., x_n)$. When we are able to

evaluate *[J]* and *[H]* in explicit forms, we will prefer the white-box method. The white box is for sure often more efficient and accurate to evaluate *[J]* and *[H]*. In physical problems based on physical laws such as fluid dynamics where turbulence can be the physical outcome, *[J]* and *[H]* have to be evaluated with specific discretization as a white box to be even stable (such as upwinding), where the black box will need specific ways of computation.

You may ask why we need to know *[J]* and *[H]*, which are $n \times n$ matrices? It seems that we make the original vector problem larger or more complex. There are three main reasons:

1. *[J]* and *[H]* are often sparse matrices, especially when they are derived from physical problems. The sparsity comes from the nearest neighbors, or the finite connectivity in circuits, or the finite number of coupling variables in social science.
2. This is often *the best* we can do, as other methods are not stable or even more computationally expensive (such as the bisection method).
3. When we use "simpler" or "computationally cheaper" search methods, Eqs. (2) – (5) serve as sound theoretical base to know how good our present approximate method is.

It is surely not strange that we ask for the local gradient information during a computational problem, as this is called the "small signal" analysis in circuits and "margin" analysis in economics.

2.3 Examples of $n = 1$ of the root finding problem

We will work out the detailed solution procedure for the solution of $f(x) = x - \cos(x) = 0$ for illustration, where we know there is a solution in $(0, \pi/2)$ from Fig. 2, but we also know that this transcendental equation does not have closed-form solution. As the direct solution is now not possible, we have to adopt an iterative scheme. We will also define all of the formal terms where the meaning is clear with the simple example here. Surely, we can use the bisection method in 1D to search the region of $(0, \pi/2)$, which will be sufficiently efficient, and k digits of precision will need $k \cdot \log_2 10$ steps of search. When we can converge with the speed of the bisection method, we call this the "**linear convergence**" (it is actually exponential, but the improvement is proportional to $\exp(\Delta x)$ instead of $\exp(\Delta x^2)$).

In general, in an iterative method, we make an initial guess of $x^{(0)}$, and use it to evaluate $\Delta x^{(0)}$, and make $x^{(1)} = x^{(0)} + \Delta x^{(0)}$, and the process goes on until:

$$\lim_{k \to \infty} f\left(x^{(k)}\right) \to 0 \qquad (7)$$

During this search, when Eq. (7), as the **absolute residue**, approaches 0, it is required that:

$$\lim_{k \to \infty} \Delta x^{(k)} \to 0 \qquad (8)$$

Equation (8) is called the **relative residue**. Notice that Eq. (7) implies Eq. (8), but not vice versa. Equation (7) is the best criteria for checking convergence, although we use Eq. (8) often for convenience. In multi-variable cases, Eqs. (7) and (8) will become:

$$\lim_{k \to \infty} \left\| \bar{f}\left(\bar{x}^{(k)}\right) \right\|_2 \to 0 \qquad (9)$$

$$\lim_{k\to\infty} \frac{\left\|\Delta\vec{x}^{(k)}\right\|_2}{\left\|\vec{x}^{(k)}\right\|_2} \to 0 \qquad (10)$$

We can see here why Eq. (10) is sometime more convenient to check in multi-variable cases, as it is easier to normalize when there are round-off errors involved: it is more difficult to say how small is small enough for un-normalized numbers.

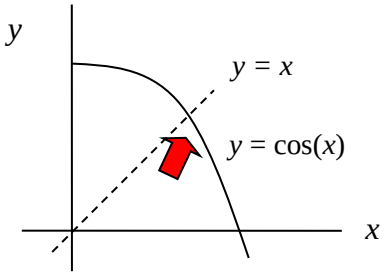


Fig. 2. Intersection of $y = x$ and $y = \cos(x)$ gives the root for $x - \cos(x) = 0$.

As mentioned before, the bisection method is only good for very small numbers of degrees of freedom due to the large domain of searching with large n. How can we search more effectively? Or equivalently, how can we find Δx that can converge faster than bisection? The answer is similar in the local analysis. The bisection method only uses the sign of $f(x^{(k)})$. Can we use $\frac{df\left(x^{(k)}\right)}{dx}$ as well to make better Δx? For example, if $f(x)$ is reasonably linear around the present guess, we know that we should change x by the amount of $\Delta x = -f(x)/f'(x)$ for the next solution. This is the **Newton's method,** also known as the **Newton-Raphson** method, as shown in Fig. 3. We will make correction to Δx so that IF $f(x)$ is linear, it will find the solution in one step. This selection of Δx will make the convergence to be "quadratically better", as the error is now Δx^2, when we approach the solution with $\Delta x \to 0$, the error of Δx^2 goes to zero quadratically faster. If the bisection method converges exponentially as $\exp(\Delta x)$, then the second-order Newton method will converge proportionally to $\exp(\Delta x^2)$, surely with small Δx.

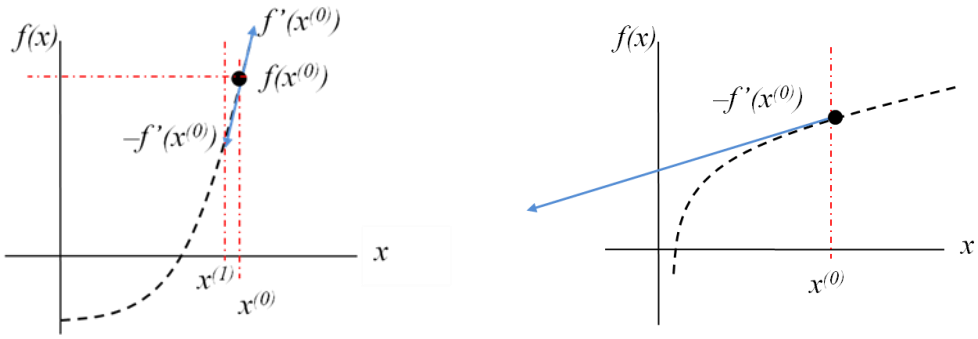


Fig. 3. Using the slope information $f'(x)$ to find $f(x) = 0$, and possible overshoot problems.

For the Newton method, the good news is, if we are close to the solution (i.e., Δx is small), all continuous equations will be close to linear, as can be seen from the Taylor series: $f(x) = f(x_0) + f'(x_0)\Delta x + O(\Delta x^2)$. The bad news is, for a highly nonlinear problem, we can "**overshoot**" the correction to bring it to a region with very large error. For example, $f(x)$ can contain e^x and not "convergent" when x is very large. Or $f(x)$ can contain $\log(x)$ and not convergent when x is close to zero.

The Newton's method can be summarized in the steps below (taking the full step without line search). As the linear term is explicitly considered, it is expected to have **quadratic convergence**.

1. Set up the variables and evaluation of $f(x)$ (and its second norm as the absolute residue) for solving x that satisfis the nonlinear equation $f(x) = 0$
2. Make an initial guess $x^{(k)}$. Notice that this is a difficult choice and has dominant influence on the convergence behavior. Unfortunately, sometime we may not know how to make a good initial guess. We will talk more about that afterwards.
3. Evaluate $f(x^{(k)})$ and its slope $f'(x^{(k)})$ or the Jacobian matrix J for the multi-variate case. Calculate the update vector $\Delta x^{(k)} = -f(x^{(k)})/f'(x^{(k)})$ or $\Delta\vec{x}^{(k)} \cong -\left[J\left(\vec{x}^{(k)}\right)\right]^{-1} \cdot \vec{f}\left(\vec{x}^{(k)}\right)$.
4. Evaluate the norm of $\|\Delta x^{(k)}\|_2$ and $\|f(x^{(k)})\|_2$. Stop if $\|\Delta x^{(k)}\|_2$ or $\|f(x^{(k)})\|_2$ < tolerance (often set between 10^{-7} to 10^{-9}).
5. Update $x^{(k+1)} = x^{(k)} + \Delta x^{(k)}$, increment k by 1, and return to Step 3 to iterate.

For example, the quadratic convergence rate is shown in Table 1.

Table 1. The linear and quadratic convergence rates.

Step size or residual	Linear convergence	Quadratic convergence
$\frac{\left\|\Delta x^{(k)}\right\|}{x^{(k)}}$ or $\left\|f\left(x^{(k)}\right)\right\|$	0.1	0.1
$\frac{\left\|\Delta x^{(k+1)}\right\|}{x^{(k+1)}}$ or $\left\|f\left(x^{(k+1)}\right)\right\|$	0.01	0.01
$\frac{\left\|\Delta x^{(k+2)}\right\|}{x^{(k=2)}}$ or $\left\|f\left(x^{(k+2)}\right)\right\|$	10^{-3}	10^{-4}
$\frac{\left\|\Delta x^{(k+3)}\right\|}{x^{(k+3)}}$ or $\left\|f\left(x^{(k+3)}\right)\right\|$	10^{-4}	10^{-8}
$\frac{\left\|\Delta x^{(k+4)}\right\|}{x^{(k+4)}}$ or $\left\|f\left(x^{(k+4)}\right)\right\|$	10^{-5}	10^{-16}

The quadratic convergence not only is faster, but is an important **validation** tool, as it shows that the Jacobian is evaluated correctly with the nonlinear equation of interest (i.e., the implementations of $f(x)$ and $f'(x)$ or J are consistent).

Example 1:

$f(x) = x - \cos x$; Initial guess: $x = 0$. We will neglect the line search here just for illustration.

$f'(x) = 1 + \sin x;\ \Delta x^{(k)} = -\left[f'\left(x^{(k)}\right)\right]^{-1} f\left(x^{(k)}\right) = -\dfrac{x^{(k)} - \cos x^{(k)}}{1 + \sin x^{(k)}}$

$x^{(0)} = 0;\ \Delta x^{(0)} = 1.$ (No line search use. If we do use, we will search $\Delta x^{(0)} = 2, 1, 0.5, 0.25, 0.125$ to see which one gives the smallest $f(x^{(1)})$.)

$x^{(1)} = 1;\ \Delta x^{(1)} = -0.24.$

$x^{(2)} = 0.76;\ \Delta x^{(2)} = \ldots.$

Hacker Practice 5.2:

Use the Newton method to solve the following nonlinear equation:

$$f(x) = e^{100x} - 1 = 0$$

$$\Delta x^{(k)} = -[f'(x^{(k)})]^{-1} f(x^{(k)})$$

Report $x^{(k)}$, $\Delta x^{(k)}$, $f(x^{(k)})$.
Make $x^{(0)} = 1$, and then recompute using $x^{(0)} = 10$. When do you observe quadratic convergence?

2.4 The Newton method in the multi-variate case

From the definition in Eq. (2), we can write the following in the vector form:

$$\vec{f}\left(\vec{x} + \Delta\vec{x}\right) - \vec{f}\left(\vec{x}\right) = \left[J\left(\vec{x}\right)\right] \cdot \Delta\vec{x} + O\left(\left\|\Delta x\right\|^2\right) \qquad (11)$$

When $\vec{f}\left(\vec{x}\right)$ is close to a linear function, we have $\vec{f}\left(\vec{x} + \Delta\vec{x}\right) = 0$ in Eq. (11). Therefore, the correction vector $\Delta\vec{x}$ in the Newton method will be:

$$\begin{aligned} \left[J\left(\vec{x}\right)\right] \cdot \Delta\vec{x} &\cong -\vec{f}\left(\vec{x}\right) \\ \Delta\vec{x} &\cong -\left[J\left(\vec{x}\right)\right]^{-1} \cdot \vec{f}\left(\vec{x}\right) \end{aligned} \qquad (12)$$

Example 2:

$$\begin{aligned} f_1 &: \quad 3x_1^2 + x_2 - 4 = 0 \\ f_2 &: \quad x_1^2 - 3x_2 + 2 = 0 \end{aligned}$$

Notice that the nonlinearity is only in x_1. We also know that there are two solutions at (1, 1) and (–1, 1).

$$J = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} \end{bmatrix} = \begin{bmatrix} 6x_1 & 1 \\ 2x_1 & -3 \end{bmatrix}$$

If we use an initial guess of (1, 0), (Notice that x_1 is already correct, and x_2, which is linear, is away from one of the true solutions), we will obtain the first step[1],

$$x^{(0)} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}; \quad \Delta x^{(0)} = -\begin{bmatrix} 6 & 1 \\ 2 & -3 \end{bmatrix}^{-1} \begin{bmatrix} -1 \\ 3 \end{bmatrix} = \frac{1}{20} \begin{bmatrix} -3 & -1 \\ -2 & 6 \end{bmatrix} \begin{bmatrix} -1 \\ 3 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

$$x^{(1)} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}; \quad \Delta x^{(1)} = -\begin{bmatrix} 6 & 1 \\ 2 & -3 \end{bmatrix}^{-1} \begin{bmatrix} 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$ Converged to the solution already.

This is of no surprise, as the problem is linear to x_2 and x_1 is right in the initial guess, so the Newton's method should arrive at the solution in one step.

If we look at a different initial guess of (2, 0), then the Newton iteration is:

$$x^{(0)} = \begin{bmatrix} 2 \\ 0 \end{bmatrix}; \quad \Delta x^{(0)} = -\begin{bmatrix} 12 & 1 \\ 4 & -3 \end{bmatrix}^{-1} \begin{bmatrix} 8 \\ 6 \end{bmatrix} = \frac{1}{40} \begin{bmatrix} -3 & -1 \\ -4 & 12 \end{bmatrix} \begin{bmatrix} 8 \\ 6 \end{bmatrix} = \begin{bmatrix} -0.75 \\ 1 \end{bmatrix}$$

$$x^{(1)} = \begin{bmatrix} 1.25 \\ 1 \end{bmatrix}; \quad \Delta x^{(1)} = \ldots$$

We can see that we still achieve the right value for x_2 in one step, but there would be several iterations needed to get to the right x_1. □

2.5 Line search in the Newton's method

Equation (12) also shows one of the most serious problems in the Newton method, as it is only true when:

1. $\|\Delta\vec{x}\|$ is sufficiently small.
2. $\vec{f}(\vec{x} + \Delta\vec{x})$ is close to zero!
3. $\Delta\vec{x}$ is NOT zero (or else we cannot make any improvement in finding the solution).
4. J^{-1} will not stretch the f vector by much, i.e., J^{-1} is not ill-conditioned.

The second criterion means that we are very close to the solution by linear approximation, which is not valid especially with bad initial guesses. To avoid instability when we are far away from the solution yet, we often opt to use a "**line search**" method to stablize the Newton method, i.e., we seek for a scalar parameter t, where

---

[1] $A = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \quad \Rightarrow \quad A^{-1} = \frac{1}{(ad - bc)} \begin{bmatrix} d & -b \\ -c & a \end{bmatrix}$.

$$\left\| \vec{f}\left(\vec{x} + t\Delta\vec{x}\right) \right\| \quad \text{is minimum for all } t \qquad (13)$$

As *t* is a scalar, we can use a bisection search efficiently. Equation (13) guarantees that each correction is a maximum improvement in the search direction of $\Delta\vec{x}$. We will talk about the line search method as well as the choice of the initial guess more when we make a similar formulation in the optimization problem.

The third criterion is problematic for the deflection point (or the saddle point) when $\Delta\vec{x}$ vanishes faster than $\vec{f}\left(\vec{x} + \Delta\vec{x}\right)$, even though $\left\| \vec{f}\left(\vec{x} + t\Delta\vec{x}\right) \right\|$ is decreasing. This method can be reasonably resolved by the line search as well by taking $t > 1$, similar to the over-relaxation in the iterative matrix solver.

Hacker Practice 5.3:

Use the Newton method with line search to solve the same nonlinear equation by making $x^{(0)} = 1$:

$$f(x) = e^{100x} - 1 = 0$$

$$\Delta x^{(k)} = -[f'(x^{(k)})]^{-1} f(x^{(k)})$$

Report $x^{(k)}$, $\Delta x^{(k)}$, $f(x^{(k)})$.
What is the change in the beginning and end of the convergence behavior?

2.6 Observation of Jacobian matrix formation

From the previous example, we can make further observation on the Jacobain matrix formation:

1. If *x* is transformed to *ax + b* with *a* and *b* being constant, or in the vector case: $\vec{y} = [A]\vec{x} + \vec{b}$ where *[A]* is a constant nondegenerate matrix and $\vec{b}$ is a constant vector, the Jacobian evaluation and the search steps will NOT change at all. The Newton method has "affine invariance"[2].
2. *[J]* can be sparse if there are many f_i's having a zero coefficient on x_j.
3. $[J]^{-1}$ will be best calculated directly or symbolically. The more accurate we can estimate J_{ij}, the better search we get (i.e., if the function is linear to that variable, we will get the correct solution in one step).
4. The **Quasi-Newton** method: If the analytical expression of J_{ij} is not available, we can evaluate J_{ij} numerically, similar to our local analysis to whatever accuracy with a small Δx_i.

$$J_{ij} \equiv \frac{\partial f_j}{\partial x_i} \cong \frac{f_j\left(x_1, x_2, ..., x_i + \Delta x_i, ..., x_n\right) - f_j\left(x_1, x_2, ..., x_i, ..., x_n\right)}{\Delta x_i} \qquad (14)$$

Here the Jacobian matrix is constructed from a local approximation, instead of evaluated directly on an analytical expression. Notice that each J_{ij} needs an evaluation of *f*, which means the formulation of the Jacobian is quite expensive, even if it is sparse. The quasi-Newton method will show quadratic

[2] Linear invariance means *x* can be transformed to *ax* without changing the system.

convergence in the basin of attraction. However, due to possible errors in the local analysis, the basin of attraction may not be the same.

After understanding the quasi-Newton method, we can ask whether we can use $\Delta x_i^{(k-1)}$ to evaluate Eq. (14)? The advantage will be one less evaluation of $\vec{f}$ by using the following approximation for the Jacobian element:

$$J_{ij} \equiv \frac{\partial f_j}{\partial x_i} \cong \frac{f_j\left(x_1^{(k)}, x_2^{(k)}, ..., x_n^{(k)}\right) - f_j\left(x_1^{(k-1)}, x_2^{(k-1)}, ..., x_n^{(k-1)}\right)}{x_i^{(k)} - x_i^{(k-1)}} \tag{15}$$

Equation (15) is called the **secant method**, but it is notoriously slow in the initial approximation due to the inaccuracy introduced in the Jacobian evaluation. Notice that the numerator in the right-hand side is the same for the same *j*, which makes the computation very cheap. The secant method most often will not show quadratic convergence, and the line-search method is usually necessary to stabilize the search.

There is one more tricky thing for the secant method. You need two initial guesses, or you need to have a guess of the "slope" together with your initial guess, so that the first Jacobian can be properly formulated. As an initial guess is already a "difficult guess" when we do not know the problem well, the guess on the slope or the second guess can be even more "wild". This is a serious drawback for the secant method. In scientific and engineering problems, the first initial guess of a dynamic equation is often the steady state or equilibrium, which we can know from the "detailed balance" principle, i.e., the time derivatives of any measurable quantity is zero at $t = 0^-$. However, we do not have a good idea about the slope at $t = 0^+$ or the second guess after perturbation.

Hacker Practice 5.4:

Use the quasi-Newton method with line search to solve the same nonlinear equation by making $x^{(0)} = 1$ and the local analysis of the Jacobian matrix by 10^{-4} perturbation:

$$f(x) = e^{100x} - 1 = 0$$

$$\Delta x^{(k)} = -\left[\frac{f(1.0001x^{(k)}) - f(x^{(k)})}{0.0001x^{(k)}}\right]^{-1} f\left(x^{(k)}\right)$$

Report $x^{(k)}$, $\Delta x^{(k)}$, $f(x^{(k)})$.
What is the convergence behavior in comparison with the Newton method with the Jacobian matrix evaluated symbolically?

2.7 Initial guess: Attractor and basin of attraction for the initial guess

The convergence behavior of the Newton's method depends heavily on the initial guess. In the multi-variate case, it is often unclear what would be an appropriate guess. Here the prior knowledge will help significantly, whether it is the previous time step or a similar case or an understanding analytically. The troublesome part is when the slope is nearly zero or *J* is nearly degenerate at the present solution. For the optimization problem, the convergence zone is termed as the **basin of attraction** as shown in Fig. 4. Notice that the Newton iteration will find the direction to decrease the objective function ONLY if the present guess is in the basin of attraction. The trajector that leads to convergence is called the attractor.

However, we can see that there can be "deflection points" where the Newton step may overshoot far away from the basin of attraction when the slope is nearly zero or *J* is nearly degenerate. The path leads to divergence is called the repellor.

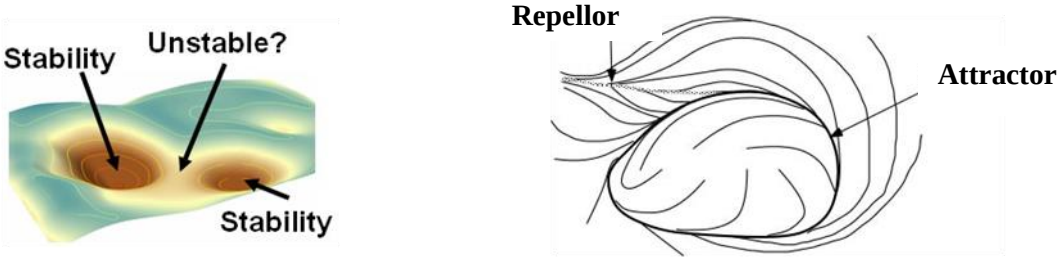


Fig. 4. Basin of attraction when the Newton method will have quadratic convergence labeled as stability. Notice that the unstable region or the deflection point has very small slope and can easily overshoot far without line search. The attractor is a trajectory leads to stable solution, while the repellor will lead away from the solution.

Appropriate initial guess for the nonlinear equation solution and the optimization problem remains an "art", as no general principle can be derived for generic problems. After we investigate the optimization problem, we will introduce a few methods to avoid the deflection points (conservative line search initially) and the step to be kicked out of the local minimum (numerical annealing or evolution). A lot of research is still going on, and let's see machine learning in the future can bring us to another level of understanding.

3. Optimization of a Scalar Function with Multiple Independent Variables.

3.1 Newton method for nonlinear optimization

We will now turn to the optimization problem. Do remember that there is a strong correspondence with the previous methods for solving the nonlinear equations. As established in Eqs. (4) and (5), we are soloving now:

$$\nabla V(x_1, x_2, ..., x_n) = \begin{pmatrix} \frac{\partial V(x_1, x_2, ..., x_n)}{\partial x_1} \\ \frac{\partial V(x_1, x_2, ..., x_n)}{\partial x_2} \\ ... \\ \frac{\partial V(x_1, x_2, ..., x_n)}{\partial x_n} \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ ... \\ 0 \end{pmatrix} \qquad (16)$$

and the Hessian matrix can be expressed as:

$$[H]=\begin{bmatrix} \frac{\partial}{\partial x_1}\left(\frac{\partial V}{\partial x_1}\right) & \frac{\partial}{\partial x_2}\left(\frac{\partial V}{\partial x_1}\right) & \cdots & \frac{\partial}{\partial x_n}\left(\frac{\partial V}{\partial x_1}\right) \\ \frac{\partial}{\partial x_1}\left(\frac{\partial V}{\partial x_2}\right) & \frac{\partial}{\partial x_2}\left(\frac{\partial V}{\partial x_2}\right) & \cdots & \frac{\partial}{\partial x_n}\left(\frac{\partial V}{\partial x_2}\right) \\ \cdots & \cdots & \cdots & \cdots \\ \frac{\partial}{\partial x_1}\left(\frac{\partial V}{\partial x_n}\right) & \frac{\partial}{\partial x_2}\left(\frac{\partial V}{\partial x_n}\right) & \cdots & \frac{\partial}{\partial x_n}\left(\frac{\partial V}{\partial x_n}\right) \end{bmatrix} \qquad (17)$$

The Newton method for searching the optimal value of *V* will be:

$$\Delta\vec{x}^{(k)} = -\underbrace{\left[H\left(\vec{x}^{(k)}\right)\right]^{-1}}_{[[J]]}\underbrace{\nabla V\left(\vec{x}^{(k)}\right)}_{f\left(\vec{x}^{(k)}\right)} \qquad (18)$$

Equation (18) simply states that the search for optimization of $V(\vec{x})$ is the search for the root finding of $\nabla V(\vec{x})=0$. We can make further observation from the Taylor expansion:

$$V(\vec{x}+\Delta\vec{x}) = V(\vec{x}) + \nabla V(\vec{x})\Delta\vec{x} + \frac{1}{2}(\Delta\vec{x})^t[H(\vec{x})]\Delta\vec{x} \qquad (19)$$

At the minimum point $\vec{x}^*$, $\nabla V(\vec{x}^*)=0$ and $V(\vec{x}^*+\Delta\vec{x})>V(\vec{x}^*)$ (i.e., $V(\vec{x}^*)$ is minimum), if and only if

$$(\Delta\vec{x})^t[H(\vec{x}^*)]\Delta\vec{x} > 0, \quad \forall\Delta\vec{x} \qquad (20)$$

We can generalize the statement in Eq. (20) for the definition of the minimum point.

- Local minimum at $\vec{x}^*$: For a given *R* > 0, if $V(\vec{x}^*+\Delta\vec{x})\geq V(\vec{x}^*) \quad \forall\|\Delta\vec{x}\|<R$
- Strict local miminum: For a given *R* > 0, if $V(\vec{x}^*+\Delta\vec{x})>V(\vec{x}^*) \quad \forall\|\Delta\vec{x}\|<R$
- Global minimum: $R\to\infty$
- The minimum is nondegenerate if *[H]* is positive definite.

Maximum can be defined vice versa.

Before ending this section, Eq. (17) has some complications in finding the second derivatives of the off-diagonal terms, if we will use the local analysis to numerically evaluate the Hessian matrix, i.e., the quasi-Newton method. For taking partial derivative twice with x_1 and x_2, you can use straightforward derivation from backward difference, then you will have an asymmetry approximation (i.e., the order of x_1 and x_2 will cause a small difference in $\partial^2 V/\partial x_1\partial x_2$):

$$\frac{\partial^2 V}{\partial x_1 \partial x_2} \cong \frac{V(\vec{x} + \Delta x_1 + \Delta x_2) - 2V(\vec{x} + \Delta x_1) + V(\vec{x})}{\Delta x_1 \Delta x_2} \qquad (21)$$

If you use the central difference, then you will have a symmetric form. It is a bit involved in the finite-difference derivation. More information can be found on the Wiki page[3].

Hacker Practice 5.5:

Use the quasi-Newton method with line search to solve the nonlinear optimization function V by making $x^{(0)} = (0, 0)$ and the local analysis of the Hessian matrix by 10^{-4} perturbation.

$$V = (3x_1^2 + x_2 - 4)^2 + (x_1^2 - 3x_2 + 2)^2$$

Report $x^{(k)}$, $\Delta x^{(k)}$, t, $f(x^{(k)})$.
Can you observe the quadratic convergence?
We know there are two local minima where $V = 0$ at (1, 1) and (–1, 1). How will you change the initial guess to get both?

3.2 Descent methods for nonlinear optimization

For finding minimum, we define a decent direction for $\vec{x}^{(k+1)} = \vec{x}^{(k)} + \Delta\vec{x}^{(k)}$ if

$$V(\vec{x}^{(k+1)}) = V(\vec{x}^{(k)} + \Delta\vec{x}^{(k)}) < V(\vec{x}^{(k)}) \qquad (22)$$

Or equivalently

$$V(\vec{x}^{(k)} + \Delta\vec{x}^{(k)}) - V(\vec{x}^{(k)}) = \nabla V(\vec{x}^{(k)}) \cdot \Delta\vec{x} < 0 \qquad (23)$$

If $\|\Delta\vec{x}^{(k)}\|$ is small, we know that the Newton direction $\Delta\vec{x}^{(k)} = -[H(\vec{x}^{(k)})]^{-1} \nabla V(\vec{x}^{(k)})$ is one of the choices of the descent directon if $[H(\vec{x})]$ is positive definite:

$$V(\vec{x}^{(k)} + \Delta\vec{x}^{(k)}) - V(\vec{x}^{(k)}) \cong \nabla V(\vec{x}^{(k)}) \cdot \Delta\vec{x} = -\nabla V(\vec{x}^{(k)}) \cdot [H(\vec{x}^{(k)})]^{-1} \nabla V(\vec{x}^{(k)}) < 0 \qquad (24)$$

Like any nonlinear iterative method, the **line-search** method will significantly stablize the search. For Eq. (23), the line search parameter t (a scalar that can be found by bisection) can be found by $V(\vec{x}^{(k)} + t\Delta\vec{x}^{(k)}) - V(\vec{x}^{(k)})$ giving the most negative number.

When the line-search method is used, we can opt to use an indentity matrix to replace $[H(\vec{x})]$. This for sure has degraded the search to be without quadratic convergence, but it saves the evaluation of the

[3] https://en.wikipedia.org/wiki/Finite_difference

Hessian matrix. We will eventually converge as the line-search method guarantees the decrease of the objective function (until it cannot be decreased further). Mathematically, we have:

$$\Delta\vec{x}^{(k)} = -t\nabla V\left(\vec{x}^{(k)}\right) = -t\frac{V\left(x_1, x_2, ..., x_i + \Delta x_i, ..., x_n\right) - V\left(x_1, x_2, ..., x_n\right)}{\Delta x_i} \tag{25}$$

Because with respect to x_i, we are descending in the local slope, we call this method the **steepest descent** or the **gradient** method. There are several variations of the gradient methods. First, the Gauss-Seidel method[4] dictates that we change one variable at a time, i.e.,

$$\Delta\vec{x}^{(k)} = -t\frac{V\left(x_1 + \Delta x_1, x_2 + \Delta x_2, ..., x_i + \Delta x_i, ..., x_n\right) - V\left(x_1 + \Delta x_1, x_2 + \Delta x_2, ..., x_i, ...x_n\right)}{\Delta x_i} \tag{26}$$

The second variation is the **conjugate gradient** (CG) method, where we make each $\Delta\vec{x}^{(k)}$ to be orthogonal to the previous steps of $\Delta\vec{x}^{(0)}$ to $\Delta\vec{x}^{(k-1)}$ by taking superposition with the previous steps. For example, for $n = 2$, if $\Delta\vec{x}^{(0)} = (1, 0)$, and the first calculation of $\Delta\vec{x}^{(1)}$ is (1, 1). The modified step in the CG method will be

$$\left(\Delta\vec{x}^{(1)}\right)_{CG} \cdot \Delta\vec{x}^{(0)} = 0 \text{ where } \left(\Delta\vec{x}^{(1)}\right)_{CG} = a\Delta\vec{x}^{(0)} + \Delta\vec{x}^{(1)} \tag{27}$$

Solving for a, we can obtain $\left(\Delta\vec{x}^{(1)}\right)_{CG}$ =(0, 1) with $a = -1$. The advantage of the CG method is that when the problem is nearly linear, we can guarantee to find the solution in less than n steps, as the correction vector would have covered the entire space with n orthogonal vectors. Notice that when V is highly nonlinear, the CG method may not find a minimum in n steps.

Hacker Practice 5.6:

Use the steepest descent method with line search to solve the nonlinear optimization function V by making $x^{(0)} = (0, 0)$ and the local analysis by $\Delta x_i = 10^{-4} \cdot x_i$ perturbation.

$$V = (3x_1^2 + x_2 - 4)^2 + (x_1^2 - 3x_2 + 2)^2$$

$$\Delta\vec{x}^{(k)} = -t\nabla V\left(\vec{x}^{(k)}\right) = -t\frac{V\left(x_1, x_2, ..., x_i + \Delta x_i, ..., x_n\right) - V\left(x_1, x_2, ..., x_n\right)}{\Delta x_i}$$

Report $x^{(k)}$, $\Delta x^{(k)}$, t, $f(x^{(k)})$.
Can you observe the quadratic convergence?
We know there are two local minima where $V = 0$ at (1, 1) and (–1, 1). How will you change the initial guess to get both?

There are two special cases for nonlinear approximation which we will investigate below. The first is the optimal parameter extraction, and the second is the spline fitting for curves or curve surfaces. Both have

[4] Notice the similar name in the iterative matrix solver. They are indeed mathematically closely related.

VERY important applications, but they share a lot of common mathematical background with our methods above.

4. Optimal parameter extraction from experiments

To describe our complex world, often a model is created with the independent variables ($x_1, x_2, \ldots x_n$) and meaningful model parameters ($a_1, a_2, \ldots, a_m$). A "scalar" model can then be generally expressed as:

$$S_{model} = S(x_1, x_2, \ldots x_n; a_1, a_2, \ldots, a_m) \qquad (28)$$

For example, S can be the current into an electronic block, where $x_1, x_2, \ldots$ are the control input voltages, and $a_1, a_2, \ldots$ are the parameters describing the blocks (e.x., the resistance and capacitance values that determines the time constants how $x_1, x_2, \ldots$ will affect S). Another possible example is S is your portfolio value, where $x_1, x_2, \ldots$ are the individual stock or fund prices, and $a_1, a_2, \ldots$ are the shares you have. The first case is nonlinear if you have transistors in the block, and the second case is mostly linear, unless you consider compound interest rates and complex coupling of fund and stock prices.

We can often make measurements of S with given values of the independent variables ($x_1, x_2, \ldots x_n$), including noises or uncertainties in the measurements.

$$S_{measured} = S(x_1, x_2, \ldots x_n) \qquad (29)$$

S can be a vector of rank l as well, but we will restrict here for S to be a scalar to be consistent with our optimization setup. The least-square parameter extraction to construct the model parameters ($a_1, a_2, \ldots, a_m$) is an optimization problem to minimize the following function V that describe the least-square fitting:

$$V = \sum_i \left(S_{i,model} - S_{i,measured}\right)^2 \text{ for all } i\text{-th instances with the same } (x_1, x_2, \ldots x_n) \qquad (30)$$

To map the parameter extraction problem to the nonlinear equation solver, Eq. (30) is equivalent to solve the set of m nonlinear equations by:

$$\frac{\partial V}{\partial a_j} = 0; \qquad \forall j = 1 \ldots m \qquad (31)$$

As V is always positive regardless of ($x_1, x_2, \ldots x_n$) and ($a_1, a_2, \ldots, a_m$), it is straightforward to derive that the Hessian matrix is positive definite and we will always reach a minimum (least in "least square"). If S is a vector, Eq. (31) will be changed to Eq. (32) with most procedures staying the same.

$$\frac{\partial \|V\|}{\partial a_j} = 0; \qquad \forall j = 1 \ldots m \qquad (32)$$

where $\|V\|$ represents taking the norm in Eq. (30). You can see that we will have the exact number of m equations for the m parameters. As V is nonlinear, even when S is linear[5], there may be more than one set of solutions ($a_1, a_2, \ldots, a_m$), but we will not have under-determined or over-determined problems. The

[5] When S is linear to x and a, this can be the conventional least-square fit of a line. When S is nonlinear to x and a, V will be even more nonlinear to x and a.

purpose of the model parameter extraction is often for estimation of S for a given (x_1, x_2, … x_n) that we do not have direct measurements. Also, $S_{measurement}$ may not be smooth, which makes taking the numerical derivative extremely noisy. However, S_{model} can often be made to be C-∞, meaning all derivatives of S are continuous. For example, x is a function of time t, we can use S for making predictions. Or in power network, we can ask what x can make S to be unstable (by definition, you can not measure the unstable situation, or in general, we do not often perform experiments on large-scale power network anyway). Another example would be a differential circuit, where the output depends on the derivative of some measurable quantities.

Example 3:

We will use a simple linear fitting to illustrate least-square fitting from the experimental observations on the given random variables. The model will now be linear, i.e., $y = ax+b$, where x and y are random variables (but they will have noises during measurements) and a and b are parameters. If we have four measured points (notice that the indices now are NOT different variables, but just different constants) of (x_0, y_0), (x_1, y_1), (x_2, y_2), and (x_3, y_3), we have the least square function as:

$$V(a,b)=(ax_0+b-y_0)^2+(ax_1+b-y_1)^2+(ax_2+b-y_2)^2+(ax_3+b-y_3)^2$$

The best parameters (a, m) that fit the four measurements will satisfy:

$$\frac{\partial V(a,b)}{\partial a}=2x_0(ax_0+b-y_0)+2x_1(ax_1+b-y_1)+2x_2(ax_2+b-y_2)+2x_0(ax_3+b-y_3)=0$$

$$\frac{\partial V(a,b)}{\partial b}=2(ax_0+b-y_0)+2(ax_1+b-y_1)+2(ax_2+b-y_2)+2(ax_3+b-y_3)=0$$

We can reorganize the previous two equations to become:

$$f_1=(x_0^2+x_1^2+x_2^2+x_3^2)a+(x_0+x_1+x_2+x_3)b-(x_0y_0+x_1y_1+x_2y_2+x_3y_3)=0$$

$$f_2=(x_0+x_1+x_2+x_3)a+4b-(y_0+y_1+y_2+y_3)=0$$

We can see that f_1 and f_2 are linear to both a (the slope) and b (the intersection to y axis). The Hessian matrix for the least-square minimum or the Jacobian matrix for the nonlinear solution to f_1 and f_2 is a constant matrix depending on the four known points:

$$[H]_V=[J]_{f1,f2}=\begin{bmatrix}\frac{\partial f_1}{\partial a} & \frac{\partial f_1}{\partial b}\\ \frac{\partial f_2}{\partial a} & \frac{\partial f_2}{\partial b}\end{bmatrix}=\begin{bmatrix}x_0^2+x_1^2+x_2^2+x_3^2 & x_0+x_1+x_2+x_3\\ x_0+x_1+x_2+x_3 & 4\end{bmatrix} \qquad (33)$$

Notice now the Hessian matrix is apparently symmetric. We can opt to solve (a, b) in one step as well, as now $[H]_V$ and $[J]_{f1,f2}$ do not depend on a or b now, and the Newton method can solve the linear problem in a single step. □

Because the least-square fitting is very important to many problems, let's use another example.

Example 4:

Many correlations can be described by the power law, i.e., $y = ax^m$, where x and y are random variables (but they will have noises during measurements) and a and m are parameters. If we have four measured points (notice that the indices now are NOT different variables, but just different constants) of (x_0, y_0), (x_1, y_1), (x_2, y_2), and (x_3, y_3), we have the least square function as:

$$V(a,m) = \left(ax_0^m - y_0\right)^2 + \left(ax_1^m - y_1\right)^2 + \left(ax_2^m - y_2\right)^2 + \left(ax_3^m - y_3\right)^2$$

The best parameters (a, m) that fit the four measurements will satisfy:

$$\frac{\partial V(a,m)}{\partial a} = 2x_0^m\left(ax_0^m - y_0\right) + 2x_1^m\left(ax_1^m - y_1\right) + 2x_2^m\left(ax_2^m - y_2\right) + 2x_3^m\left(ax_3^m - y_3\right) = 0$$

$$\frac{\partial V(a,m)}{\partial m} = 2a\cdot\left[\ln x_0\cdot x_0^m\left(ax_0^m - y_0\right) + \ln x_1\cdot x_1^m\left(ax_1^m - y_1\right) + \ln x_2\cdot x_2^m\left(ax_2^m - y_2\right) + \ln x_3\cdot x_3^m\left(ax_3^m - y_3\right)\right] = 0$$

We can reorganize the previous two equations to become:

$$f_1 = \left(x_0^{2m} + x_1^{2m} + x_2^{2m} + x_3^{2m}\right)a - \left(x_0^m y_0 + x_1^m y_1 + x_2^m y_2 + x_3^m y_3\right) = 0$$

$$f_2 = \left(\ln x_0\cdot x_0^{2m} + \ln x_1\cdot x_1^{2m} + \ln x_2\cdot x_2^{2m} + \ln x_3\cdot x_3^{2m}\right)a^2 - \left(\ln x_0\cdot x_0^m y_0 + \ln x_1\cdot x_1^m y_1 + \ln x_2\cdot x_2^m y_2 + \ln x_3\cdot x_0^m y_3\right)a = 0$$

We can see that f_1 is linear to a (the prefactor), but highly nonlinear to m (the power). The Hessian matrix for the least-square minimum or the Jacobian matrix for the nonlinear solution to f_1 and f_2 can be written as:

$$[H]_V = [J]_{f1,f2} = \begin{bmatrix} \frac{\partial f_1}{\partial a} & \frac{\partial f_1}{\partial m} \\ \frac{\partial f_2}{\partial a} & \frac{\partial f_2}{\partial m} \end{bmatrix} = \begin{bmatrix} x_0^{2m} + x_1^{2m} + x_2^{2m} + x_3^{2m} & \frac{\partial f_1}{\partial m} \\ 2a\left(\ln x_0\cdot x_0^{2m} + \ln x_1\cdot x_1^{2m} + \ln x_2\cdot x_2^{2m} + \ln x_3\cdot x_3^{2m}\right) - \left(\ln x_0\cdot x_0^m y_0 + \ln x_1\cdot x_1^m y_1 + \ln x_2\cdot x_2^m y_2 + \ln x_3\cdot x_0^m y_3\right) & \frac{\partial f_2}{\partial m} \end{bmatrix}$$

Two observations in this example. First, *[H]* does NOT appear symmetric. Symbolically, it will be required, but if you use a numerical perturbation, the purturbation order can make a difference. Second, if we have 100 points of known *(x, y)*, each entry in the Hessian matrix will have 100 terms, but the rank of Hessian remains 2 to solve for 2 parameters of (a, m). □

Hacker Practice 5.7:

Use the Quasi Newton method to perform the parameter extraction for the power-law function $y = ax^m$ with the known measurements:

x	1.0	4.5	9.0	20	74	181
y	3.0	49.4	245	1808	2.2×10^4	7.3×10^4

Use the initial guess of $a^{(0)} = 2$ and $m^{(0)} = 1$. $V(a,m) = \sum_i \left(ax_i^m - y_i\right)^2$;

$$f_1 = \left(\sum_i x_i^{2m}\right)a - \sum_i x_i^m y_i = 0;\ \ f_2 = \left(\sum_i (\ln x_i)\cdot x_i^{2m}\right)a^2 - \left(\sum (\ln x_i)\cdot x_i^m y_i\right)a = 0.$$

$$[H]_V = \begin{bmatrix} \sum_i x_i^{2m} & \dfrac{\partial f_1}{\partial m} \\ 2a\left(\sum_i (\ln x_i)\cdot x_i^{2m}\right) - \sum_i (\ln x_i)\cdot x_i^m y_i & \dfrac{\partial f_2}{\partial m} \end{bmatrix}$$

5. Spline fitting

We will look at another important optimization problem for geometrical fitting. This is actually a problem people already encountered in the ancient time when a curve surface had to be made for canoes or boats. Before we start, we will first look at a less useful case of polynomial fitting in 2D. However, the general solution to the polynomial fitting is useful and intuitive.

5.1 Polynomial fitting[6] in 2D

If we know *d+1* points in the *x-y* plane,

$$y_k = f(x_k), \text{ or } (x_k, y_k) \text{ for } k = 0, 1, 2, \ldots, d. \qquad (34)$$

where $x_i \neq x_j$ if $i \neq j$. There is a polynomial function of order *d* that can pass through these *d+1* points:

$$p(x) = p_0 + p_1 x + p_2 x^2 + \ldots + p_d x^d \text{ where } p(x_k) = y_k \qquad (35)$$

We can express $p(x_k) = y_k$ in the matrix form as:

$$[V]\vec{p} = \vec{y} \qquad (36)$$

where[7] [*V*] is called the Vandermonde matrix

---

[6] Polynomials are popular functions for intuition. The other very useful fitting function is sin(*x*) and cos(*x*), which will lead to Discrete Fourier Transform. As DFT is covered thoroughly in signal processing, we will not spend time to treat it in this class.

$$[V]=\begin{bmatrix}1 & x_0 & \dots & x_0^d\\ 1 & x_1 & \dots & x_1^d\\ \dots & \dots & \dots & \dots\\ 1 & x_d & \dots & x_d^d\end{bmatrix} \qquad \vec{p}=\begin{bmatrix}p_0\\ p_1\\ \dots\\ p_d\end{bmatrix} \qquad \vec{y}=\begin{bmatrix}y_0\\ y_1\\ \dots\\ y_d\end{bmatrix} \tag{37}$$

We know [*V*] of rank *d+1* is always nondegenerate, and there is always a solution to ($p_0, p_1, \dots, p_d$). Instead of giving a proof, we will just give an explicit form of *p(x)* below.

Definte the Lagrange polynomial as:

$$l_k(x)=\frac{\prod_{j\neq k}(x-x_j)}{\prod_{\substack{j\neq k\\ j=0,1,\dots d}}(x_k-x_j)} \tag{38}$$

For example, for $d = 2$, $x_0 = 0$, $x_1 = 2$, $x_2 = 3$, we have the first two Lagrange polynomials as:

$$l_0(x)=\frac{(x-x_1)(x-x_2)}{(x_0-x_1)(x_0-x_2)}=\frac{1}{6}(x^2-5x+6); \qquad l_1(x)=\frac{(x-x_0)(x-x_2)}{(x_1-x_0)(x_1-x_2)}=x^2-3x$$

It is straightforward to observe:

$$l_k(x_j)=\delta_{jk}=\begin{cases}1 & j=k\\ 0 & j\neq k\end{cases} \tag{39}$$

We can write down the polynomials that pass through the *d+1* points as:

$$p(x)=\sum_{k=0}^{d} y_k l_k(x) \tag{40}$$

Since we have the closed form for *p(x)*, the Vandermonde matrix must be nondegenerate. The polynomial fitting is known to have a lot of oscillations, especially when the order is high. Therefore, they are not practical in engineering applications. However, it is a useful intuitive understanding.

5.2 Fixed-point spline fitting

When we (including the ancient people) need to draw a line, we use a ruler. We also have compass for circles and protractor for angles. However, how can we construct a smooth curve when we know the fixed points it has to pass (these fixed points are called the ducks or anchors or nails or knots in various applications). As mentioned, the polynomial function is not practical due to the spurious oscillations. Can we construct piece-wise cubic functions that are "smooth"? These are called splines, where in ancient time thin sticks made of flexible wood or bamboos. When they have to pass some fixed ducks,

[7] Sorry that I used *V* here again, which may be confused with the *V* objective function in the optimization problem. However, *V* is called the Vandermonde matrix, and often has the symbol *V*.

these splines will become piecewise cubic functions that minimize the elastic energy, which makes the function and 1st and 2nd derivatives continuous. Spline curves are very useful not only in drawing curves for vehicular contours, but also a fundamental part for computer graphics and animation.

Let a spline curve S which has $d+1$ fixed points it has to pass through between $[a, b]$:

$$a = x_0 < x_1 < x_2 < \ldots < x_{d-1} < x_d = b \qquad (41)$$

There are d intervals where we need d piecewise polynomials to describe S:

$$\begin{aligned} &S(x) = p_1(x); \quad x_0 \le x < x_1 \\ &S(x) = p_2(x); \quad x_1 \le x < x_2 \\ &\ldots \\ &S(x) = p_d(x); \quad x_{d-1} \le x < x_d \end{aligned} \qquad (42)$$

If we hope that $S(x)$ is smooth to the $n-1$ derivative, we will consturct a n-degree spline curves:

$$p_i^{(j)}(x_i) = p_{i+1}^{(j)}(x_i) \text{ for } i = 1,.., d-1; j = 0,\ldots, n-1 \qquad (43)$$

Equation (43) is a general definition of spline curves, and $p_i(x)$ is sometime called the **basis spline**, or b-spline. The most useful and common spline curves have $n = 3$, i.e., cubic splines, which will pass through all the ducks and are C-2 continuous everywhere, especially at the ducks!

5.2.1 Normalization in an interval

To simplify the expression of $S(x)$ in Eq. (42), we will first normalize x for the cubic function $q_i(x)$ in the i-th interval beween (x_{i-1}, y_{i-1}) and (x_i, y_i) by: (the general polynomial $p_i(x)$ is now a cubic $q_i(x)$)

$$\begin{aligned} &q_i(x) \equiv (1-t)y_{i-1} + ty_i + t(1-t)(a_i(1-t) + b_i(t)) \\ &t = \frac{x - x_{i-1}}{x_i - x_{i-1}}; \quad a_i = k_{i-1}(x_i - x_{i-1}) - (y_i - y_{i-1}); \quad b_i = -k_i(x_i - x_{i-1}) + (y_i - y_{i-1}); \end{aligned} \qquad (44)$$

where from the continuity of the function and the first derivative, we know:

$$\begin{aligned} &q_i(x_i) = q_{i+1}(x_i) = y_i; \quad q_{i-1}(x_{i-1}) = q_i(x_{i-1}) = y_{i-1}; \\ &q_i'(x_i) = q_{i+1}'(x_i) = k_i; \quad q_{i-1}'(x_{i-1}) = q_i'(x_{i-1}) = k_{i-1}; \end{aligned} \qquad (45)$$

We can see how t is a normalized form in the interval of (x_i, y_i) for $q_i(x)$ as when $t = 0$, $x = x_{i-1}$ and when $t = 1$, $x = x_i$. The two constants a_i and b_i are used to express the difference of the linear projection from both ends. Note that we know the values of (x_i, y_i), but not k_i. For $q_i(x)$ as a cubic polynomial, if we know the point values and the derivatives at both ends (4 conditions), then $q_i(x)$ will have all four of its coefficients determined, i.e., as long as we determine all k_i, for $i = 1,\ldots,d-1$, all $q_i(x)$ will be known as the piecewise cubic function to represent $S(x)$.

5.2.2 Use the unknown first derative to guarantee continuous second derivative

How should k_i be determined (there are $d+1$ unknown k_i)? We will use it to match the second derivative at all of the interior anchor points, i.e.,

$$q_i''(x_i) = q_{i+1}''(x_i) \text{ for } i = 1,\ldots d-1 \qquad (46)$$

There are only $d-1$ conditions in Eq. (46). We will need two more conditions to determine all k_i. The most popular choice corresponding to the original use of spline curves in vehicle contours is that $S(x)$ becomes straight lines outside $[x_0, x_d]$, i.e.,

$$q_i''(x_0) = q_{i+1}''(x_d) = 0 \qquad (47)$$

It is also possible to pin k_0 and k_d directly, but that will make $S(x)$ outside of $[x_0, x_d]$ to be unconstrained cubic functions. We will use Eqs. (46) and (47) to express the set of equations to determine all k_i, and thus all coefficients of $q_i(x)$.

$$\begin{aligned}
&\frac{k_{i-1}+2k_i}{x_i-x_{i-1}}+\frac{2k_i+k_{i+1}}{x_{i+1}-x_i}=3\left(\frac{y_i-y_{i-1}}{(x_i-x_{i-1})^2}+\frac{y_{i+1}-y_i}{(x_{i+1}-x_i)^2}\right)\\
&q''(x_0)=2\cdot\frac{3(y_1-y_0)-(k_1+2k_0)(x_1-x_0)}{(x_1-x_0)^2}=0\\
&q''(x_d)=-2\cdot\frac{3(y_d-y_{d-1})-(k_d+2k_{d-1})(x_d-x_{d-1})}{(x_d-x_{d-1})^2}=0
\end{aligned} \qquad (48)$$

Equation (48) will formulate a band matrix to solve all k_i:

$$\begin{bmatrix} a_{11} & a_{12} & \ldots & 0 \\ a_{21} & a_{22} & \ldots & 0 \\ 0 & \ldots & \ldots & \ldots \\ 0 & \ldots & a_{d-1d} & a_{dd} \end{bmatrix} \begin{bmatrix} k_0 \\ k_1 \\ \ldots \\ k_d \end{bmatrix} = \begin{bmatrix} \ldots \\ \ldots \\ \ldots \\ \ldots \end{bmatrix} \qquad (49)$$

Band matrices such as the tridiagonal matrix above can be solved efficiently[8], as you have only nearest neighbor coupling.

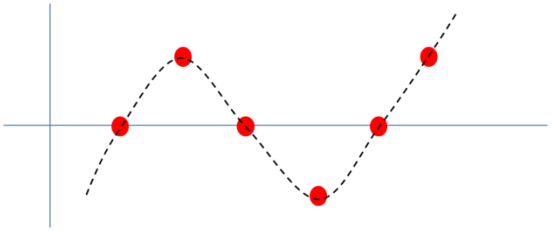
[8] https://en.wikipedia.org/wiki/Band_matrix

Hacker Practice 5.8:

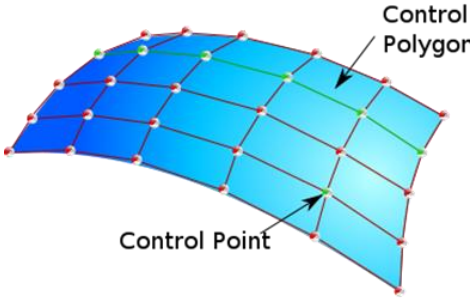
Construct a B-Spline curve with the following anchor points

x	1.0	2.0	3.0	4.0	5.0	6.0
y	0.0	1.0	0.0	–1.0	0.0	1.0

Define *S* as piecewise b-spline functions, and if you can, plot it out!



5.3 Further topics on spline functions



The first extension is the least-square fitting of spline curves with the control points instead of anchor points. This is sometime called the "smoothing splines"[9] as well. Unlike the anchor (knot) points where $S(x)$ has to pass through, the control points are where the piecewise cubic functions are separated, although $S(x)$ is C-2 continuous everywhere. For examples, we may have a lot of experimental points from an image, and we ask the questions what the coefficients of $q_i(x)$ are to formulate least-square fitting with the experimental points. The procedure to find those coefficients is very similar to least-square parameter extraction. The control points are often assigned by some known features of $S(x)$, or adaptively by the trends in the experimental points.

The second extension is to 3D, i.e., instead of finding a spline curve, we are finding the spline surface that passes through anchor points, or the smoothing spline surface that gives the least-square fitting. Combining with all spline features (C-2 continuity, anchor or control points, etc.), we can now see how NURBS (non-uniform rational b-spline) are used universally in mechanical designs, animation and image processing. It actually deserves its own course as well.

6. Monte Carlo Sampling

6.1 A short history of Monte Carlo sampling

Many of the nonlinear optimization problems need an "initial guess" to start the search. What can be that good initial start? Often the parameter space is so large that it is impossible even for very crude systematic sampling. For example, we have 30 parameters in the least-square fitting problem of Eq. (30), even four possible values of 30 parameters will span to an initial choice of 4^{30}, which is about 10^{18}!!! In many card and board games (including go and bridge), the possible sequence forms a factorial that is too large to search by computers as well. Say in a solitaire card game, the sequence of 52 cards is 52! ~ 10^{68}!!! This is beyond any number that a computer can compute directly.

[9] https://en.wikipedia.org/wiki/Smoothing_spline

How can we sample a huge space and have an idea of what has been covered? This is a problem that has puzzled physicists and mathematians for a long time without a solution. In the Manhattan project, one of the renown mathematians, Stanislaw Ulam was in charge to investigate the reliability problem of the nuclear reactor, but cannot have a satisfying answer due to the huge parametrical space. When he was confined in the hospital playing solitaire[10], he came up with the idea of statistical sampling. After a discussion with John von Neumann[11] who was also on the Manhattan project, they gave the solution a code name **Monte Carlo** (yes, all projects in the Manhattan project have code name for confidentiality).

6.2 Large number theory for Monte Carlo sampling

Let v be a random variable (now v is just a scalar, but later on it can be generalized to a vector with a large rank) whose distribution density function $p(v)$ is **unknown**. We can denote $E(v) = A$ and $\text{var}(v) = \sigma^2$, which we do not know their values either. A Monte Carlo sampling of v is represented by a group of N random sampling of v_k by defining:

$$\hat{A}_N = \frac{1}{N}\sum_{k=1}^{N} v_k; \qquad R_N = \hat{A}_N - A \tag{50}$$

where R_N is called the **bias** of the present estimator of N sampling. As v_k is random, we can have the following conclusions by the **central limit theorem** and the **large-number theorem**:

1. $E(R_N) = 0$, i.e., errors of v_k is random and equally above and below $\hat{A}_N$. The choice of these (v_1, v_2, …, v_N) that satisfies this property is called a consistent estimator.
2. $\text{var}(R_N) = \sigma^2/N$, i.e., when the expected variance will become smaller with increasing N.
3. With $N \to \infty$, R_N approaches a standard normal distribution function $\mathcal{N}(0,1)$ with zero mean and unit standard deviation:

$$p_{RN}(v) \cong \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{v^2}{2}\right) = \mathcal{N}(0,1) \tag{51}$$

These three criteria told us that even we do not know A, σ or $p(v)$, when we have N samples (or test measurements), we can estimate A by $E(R_N) = 0$, estimate σ^2 by $\text{var}(R_N) = \sigma^2/N$, and know how far we are by Eq. (51), or equivalently having an idea of the error bar. We can think that the distribution of R_N is approximately the same as $\sigma Z/\sqrt{N}$ where $Z \sim \mathcal{N}(0,1)$, i.e., we can think the error (or the bias) in the Monte Carlo testing is of the order of $1/\sqrt{N}$ with a prefactor σ (the variance of the problem under study)!

This shows the power and limitation of the Monte Carlo simulation. The power is that as long as we can do N testings, we can tackle a problem we know little about its behavior. The limitation is the statistical error is proportional to $1/\sqrt{N}$, i.e., making N to be four times larger can only improve the accuracy by 2 times, worse than even bisection, not to mention any other higher-order approximation method. We can

[10] Hospitalized physicists and mathematicians had solved many problems, strangely enough. I guess we can be most creative when we are away from daily routines and worries.

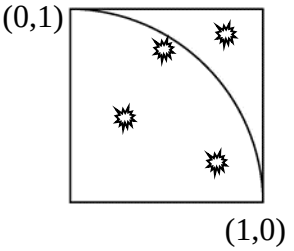
[11] John von Neumann is also the father of modern computers, computer virus, game theory and many other things. When you visit Princeton next time, remember to check his story, not just Einstein!

indeed improve the way we choose samples to make better accuracy improvement. This is generally called the **variance reduction** in Monte Carlo, such as importance sampling, antithetic variates, control variates and statistical amplification.

Program Practice 5.9: Calculating π by Monte Carlo

We know the area of a quarter circle is π/4 (and assume that we do not know the value of π for now). We can transform the calculation into a statistical sampling by estimating the probability of random distributions in a square to be within the circle. The pseudo code can be:

```
double x[i] = random( );
double y[i] = random( );
integer count = 0;
for i = 1, N; i++
      if ( x[i]*x[i] + y[i]*y[i] < 1 ) count++;
double pi = 4*count/N;
```



Now observe your error in estimation for $N = 10, 100, , \ldots 10^6$.
If you have your power-law fitting still, plot out the N vs. error.

From the previous exercise, we can make a few observations:

1. To calculate π correctly, x and y need to be uniformly distributed between (0, 1). Any bias in x and y will become a bias in the estimate of π. You can see them as distortion in the geometry!
2. We can enlarge the search domain without changing the answer. More random points will be wasted. This means the choice of sampling can affect how fast the convergence will be (the prefactor σ we discussed before).
3. The precision of π will be proportional to $\sqrt{N}$ from our previous theory.

6.3 Random number generation with uniform distribution

How do we get a uniformly distributed random number between (0, 1)? C/C++ provides two functions: rand() and random(), the first of which is an older version with more "predictable" results. The ideal random() will have the following properties:

1. Uniformly distributed between [0, 1].
2. Mean = 1; Variance = 1/12.
3. Any segment of sequence will have no correlation with the other.
4. The sequence is unpredictable, or the sequence has no memory effect.

C/C++ uses a pseudo-random number generation. First an integer is chosen (often the program counter is used as it is accessible to all routines and difficult to predict, or a number in a big selection taken sequentially). The pseudo-random number is then generated by (where a, c and m are large prime numbers):

```
x[i+1] = (a*x[i] + c) mod m;
return double x[i+1]/m;
```

The return random number can usually satisfy the first two criteria easily, but the sequence is predictable and has a fixed cycle. True random numbers that satisfy all four criteria have to be generated from hardware or from true noises instead of from a software algorithm[12]. Hardware noises that can be employed including:

- Thermal noises (vulnerable to nitrogen spraying),
- Single-particle noises (aka random telegraph noises, which may not have a uniform spectrum)
- Quantum noises (based on uncertainty principles).

6.4 Random number generation in arbitrary distribution

For many sampling problems, we may need a random number that follows a given distribution, instead of uniformly distributed. For example, when random numbers are used to sample a gas molecule velocity, we know that *v* would follow the thermodynamic law and has a Boltzmann distribution. When we use the random number to sample the electron energy in a solid, we know that the distribution will be a Fermi function. Transformation from a uniformly distributed variable to another distribution function is a straightforward mathematical procedure. Let's denote the probability density function of a random variable *v* as *p(v)*, and the corresponding cumulative distribution function as *F(v)*, where

$$F(v) = \int_{-\infty}^{v} p(x)dx \qquad (52)$$

From elementary probability, we know the follow properties of *p(v)* and *F(v)*.

1. $0 \le p(v), F(v) \le 1; \quad \forall v$
2. $\lim_{v \to -\infty} F(v) = 0; \quad \lim_{v \to \infty} F(v) = 1;$
3. *F(v)* is a monotonically increasing function.

When we match the culmulative distribution *F(v)* to that of a uniformly distributed random variable, we can then use random() to generate *p(v)*, as shown in Fig. 5(a) below. The steps are:

1. Call random() to obtain a random number *u* between [0, 1].
2. We know that $F(u) = u$.
3. Find *v* that gives $F(v) = u$, or $v = F^{-1}(u)$. The random variable *v* will now follow *p(v)*.

[12] John von Neumann: "Anyone who considers arithmetical methods of producing random digits is, of course, in a state of sin. For, as has been pointed out several times, there is no such thing as a random number — there are only methods to produce random numbers, and a strict arithmetic procedure of course is not such a method."

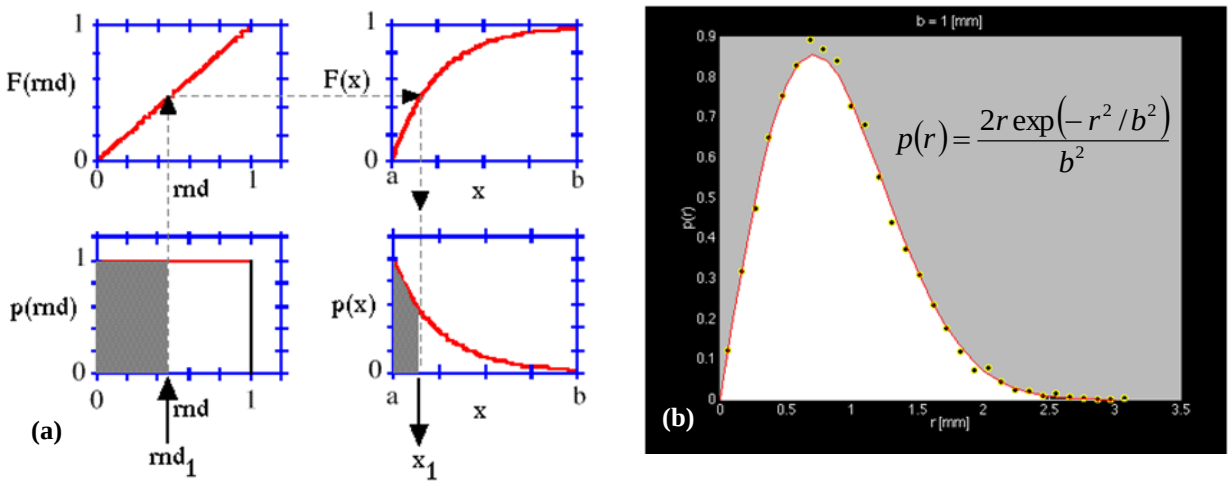


Fig. 5. (a) Transformation of a uniformly distributed random variable to an arbitrary distribution function *p(x)* by matching *F(x)*. (b) Generation of a Maxwell distribution from the histogram of such transformation. 10,000 random points are generated from a uniform distribution and then transformed by *F(x)*.

This makes sense, as when *p(v)* is very small, it would correspond to a flat region in *F(v)*, which will have a much lower probability to match with *u*.

Program Practice 5.10: To generate *v* with distribution function of *p(v)* from `random()` from known closed-form culmulative distribution function.

For a distribution function of $p(v)$ $\begin{cases} = \lambda e^{-\lambda v} & v \geq 0 \\ = 0 & v < 0 \end{cases}$, we will generate the random variable *v* from the uniformly distributed *u* between (0, 1). Use $\lambda = 0.2$.

We know that $F(v) = \int_0^v \lambda e^{-\lambda x} dx = 1 - e^{-\lambda v} = u$.

Thus, $v = -\frac{1}{\lambda}\ln(1-u)$. Check: *v* will be between (0, ∞).

Generate 1,000 instances of *v* and sort the vector *v*. Plot *p(v)* and *F(v)* for $0 \leq v < 10$ with a bin size of 0.5.

Beside visualization, is there another way for verification?

Example 6: No closed-form culmulative function:

For a distibution function that is a standand norma: $p(v) = \frac{1}{\sqrt{2\pi}}\exp\left(-\frac{v^2}{2}\right)$, we will need to generate the random variable *v* from the uniformly distributed *u* between (0, 1).

We know that $F(v) = \int_{-\infty}^{v} p(x)dx$ does not have a closed form, but we can use a table lookup or a solution to find $F^{-1}(u) = v$. □

In summary, the Monte Carlo sampling is to generate v from the random number function with the knowledge of $F(v)$. We do not need to know the closed form of $F(v)$, but just a way to **compute** $F^{-1}(u)$ from formula, table lookup or rejection rules. After obtaining N samples of v, we can use them to understand the moment, the physical phenomena governed by v, or simply to evaluate A (1st moment) and σ (2nd moment). As long as N is sufficiently large, we know our N samples will give us a reasonable estimate, as R_N will approach a standard normal distribution function. From Example 7, we know that the **confidence interval** for a Monte Carlo sampling will be:

$$\left[\hat{A}_N - \frac{\sigma}{\sqrt{N}}, \hat{A}_N + \frac{\sigma}{\sqrt{N}}\right] \qquad (53)$$

where we will be 67% confident according to the standard normal distribution. To be 99.7% confident with the 3σ deviation, the sampling will need to be in the interval of $\left[\hat{A}_N - \frac{3\sigma}{\sqrt{N}}, \hat{A}_N + \frac{3\sigma}{\sqrt{N}}\right]$.

For optimization on parameter extraction, when we need an initial guess in a large parameter space (say m= 20 in Eq. (32)), we will use 20 uniformly generated random numbers to be transformed to the known or guessed distribution function of each parameter. That would serve as one initial guess. We can repeat the process for N = 200 times to obtain a reasonable confidence that our minimal can be close to the global minimum. The quality of the initial guess (or the coverage of the confidence interval) will depend on our understanding of the distribution function of each parameter.

6.5 Simulated annealing

Finally, we can use the random number to the optimization search process directly. The most famous example is the **simulated annealing**[13], which resembles the natural process of relaxation to the energy minimum. For example, we know that diamond is the lowest-possible energy form for the group of carbon atoms. However, in many natural scenarios, the group of carbon atoms will just become a local minimum of soot (such as the product of burning, where the product is quenched quickly). To form diamond, the nature would have a high temperature first to allow the atoms to go to their preferred position (actually a high pressure is needed as well, so that they do not go too wildly), and then gradually cool down to formulate diamond.

During the minimization search process, we will adopt the simulated annealing as follows:

1. A correction step is determined by the Newton method or the steepest descent method,
2. Use the line search method to determine t for the most improvement possible in this step
3. When no improvement can be found in all searched t, instead of declaring the minimum has been found, we will take the step with the size of t anyway according to the probability function:

[13] Simulated annealing is only formally formulated in the early 1980s, nearly 30 years after the Monte Carlo method has been well established.

$$p(t) = \exp\left(-\frac{\Delta V}{T}\right) \text{ where } \Delta V \equiv \left\| V\left(\vec{x}^{(k)} + t\Delta\vec{x}^{(k)}\right) - V\left(\vec{x}^{(k)}\right)\right\|_2 . \quad (54)$$

The value of *t* will be determined by the Monte Carlo sampling with the distribution function *p(t)*. As there is no possible improvement in this step $\Delta V > 0$ and $\Delta V \in R(0,1)$. There is a temperature parameter *T* in Eq. (54) analogous to the natural annealing process. When *T* is very large, large penalty on ΔV has a higher probability to be taken. When *T* is nearly zero, only very small ΔV will have reasonable probability. For a reasonable search of the global minimum, *T* can be initially large, and then gradually becomes small during the final quenching process.

Example 7: The traveling salesman trip mileage minimization problem
A classical minimization problem is the traveling salesman. He needs to visit each city once in the 20 cities with known distance between any two cities. As there are 20! possibilities, an exhaustive search is computationally too expensive. We can start with an initial guess of city sequence, and start swapping pairs of cities by the simulated annealing. Initially we can set *T* = 100 miles, and gradually decrease *T* to 1 mile. We will achieve a reasonable minimum by just 200 to 500 Monte Carlo sampling of swapping city pairs.

Simulated annealing has been broadly applied to problems that have very large parameter space and possibly many local minima (aka, mixed determined or ambiguity). For many problems such as chess, the strategy of setting *T* can be learned from the existing examples to reduce the searching time. This machine learning strategy is for sure left to students who will take the specific course in artificial intelligence or machine learning.