ECE 4960: Computational and Software Engineering

Spring 2017

Note 3: Approximation in Local Analysis

Reading Assignments:

1. Chap. 3, D. Bindel and J. Goodman, *Principles of Scientific Computing*, 2009.
2. Chap. 3, B. Einarsson, Ed., *Accuracy and Reliability in Scientific Computing*, SIAM 2005. (Light reading for the exception case in integration approximation).

1. Class logistics

- Programming Assisgnment 1 proposal
- Reading review 2 (will be multiple choice questions on Blackboard)

2. Overview of differentiation, integration, extrapolation and intrapolation

It is often difficult to observe global behavior (weather, experiment, commerce, etc.) because our observation and measurement often have a scope and precision in space and time. It is however often critical to know the behavior between the measurements (interpolation or integration to obtain the mean value) or beyond the measurements (extrapolation or differentiation to obtain the slope or trends). "Local analysis", which investigates the behavior between and beyond the known points, is one of the most common computing practices!!! When we write programs from the local analysis to deal with differentiation and integration, how can we obtain the slope or mean values most accurately with the least computing effort? Of the same importance, for the approximation of differentiation and integration we use, how can we put an estimate on the error and how does this estimated error depend on the local resolution and the approximation formula (aka, error estimation)? How much effort is needed to improve the accuracy (aka, error adaptation)? **Error estimation** (based on the present solution and residual) and **error adaptation** (improvement in approximation precision or methods) are the central topics whenever we attempt representing the real world with computer simulation!

We will first limit ourselves to the polynomial methods for differentiation, mostly focused on the Taylor series to build up concepts and error estimation. We will illustrate how orders of approximation interact with the resolution of the local information. Eventually in formal finite-difference and finite-element methods, this will become the *hp* adaptivity (*h* for resolution enhancement and *p* for polynomial order enhancement). Intuitive error estimation such as Richardson extrapolation will be introduced. We will then generalize the polynomial method in Taylor and Legendre series for integration as in the case of **Gaussian quadrature**. We will then use special cases to illustrate that there is further ambiguity in high-order oscillation and insufficiency in funtional singularity. We will not be able to treat the general approximation cases here, but should be able to develop a good framework to look further, and more importantly, to guide your robust programming.

3. Differentiation and the Talyor series

The most straightforward theory for local analysis is based on the Taylor expansion. We will denote the approximation of a function A as $\hat{A}$. Within a resolution limit or step size h, the appromixation $\hat{A}$ is

consistent if $\hat{A} \to A$ as $h \to 0$. In the sampling theory for space and time, h can be the grid spacing or the time steps. You probably have learned before about the two approximations of finite difference for differentiation of $A = f'(x)$: first-order forward/back difference and second-order central difference:

$$f'(x) = \frac{f(x+h) - f(x)}{h} + O(h) \qquad (1)$$

$$f'(x) = \frac{f(x+h) - f(x-h)}{2h} + O(h^2) \qquad (2)$$

These two approximations are both consistent, as they will approach the true value of the differentiation when $h \to 0$. When h is small, $h^2 << h$, which implies Eq. (2) approaches consistency much faster than Eq. (1). However, Eq. (2) achieves this second-order approximation by the requirement of homogeneous spacing of the backward and forward points, which does not happen often in practical sampling. We will now derive a general method by Taylor series to evaluate the order of approximation in arbitrary sampling. By using Taylor series, we are constrained to $f(x)$ which behaves similarly to the polynomial function in the vicinity of x, which is for sure not valid around a nearly singular point that diverges when $h \to 0$, such as $f(x) = \log(x)$ at $x = 0$.

Before we go into details of local approximation, we need to remember that $h \to 0$ has another practical limit. From the last chapter, we know that when $h \to 0$, the finite precision of floating point representation will be problematic for "round off" errors, even when underflow is used carefully. Truncation errors (as defined by the finite order of approximation) and round-off errors (as defined by the round-off from the finite number of bits in the mantissa) can interplay each other for realistic problems. Therefore, either Eqs. (1) or (2) cannot be considered exactly "consistent" with finite-precision implementation. This is expressed in Fig. 1 where the X axis is h in a typical simple single-variable case. We can see that we need to stay away from the round-off errors, as there is no "cure" except to use floating-number representations with even more precision bits.

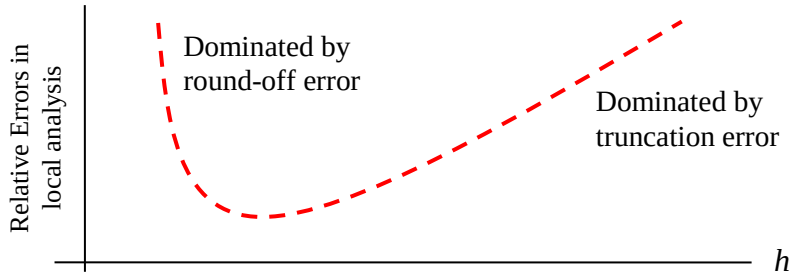

Fig. 1. Interplay between the truncation and round-off errors in local analysis.

Hacker Practice 3.1:

For $f(x) = x^2$, we know the exact $f'(x=1) = 2$. Use Eq. (1) to estimate $f'(x=1)$ varying the value of h from 0.1 to 10^{-18} to observe the relative error in calculating $f'(x)$. Repeat with $f(x) = x^2 + 10^8$. To appreciate the order of approximation, repeat the above by using Eq. (2).

3.1 Generalized Taylor approximation for differntiation

Assume that in addition to $f(x)$, we have sampling at $f(x + h_1)$ and $f(x+h_2)$. We call x the base point. We know nothing about $f(x)$ yet except a few sampling point around x, which is thus called the "**local analysis**". Notice that h_1 and h_2 can be positive or negative, but they are small for the region of interest with $h_1, h_2 \cong O(h)$ and $h_1 \neq h_2$. We are interested in approximating the differentiation of $f'(x)$. From the Taylor series, we know:

$$f(x+h_1) = f(x) + h_1 \cdot f'(x) + \frac{1}{2} h_1^2 f''(x) + O(h^3) \qquad (3)$$

$$f(x+h_2) = f(x) + h_2 \cdot f'(x) + \frac{1}{2} h_2^2 f''(x) + O(h^3) \qquad (4)$$

Notice that we write $O(h^3)$ in Eqs. (3) and (4) as all terms with h^3 or higher polynomials are truncated. When we write down $f'(x)$ by deviding h from the functional difference of $f(x)$, $O(h^3)$ will become $O(h^2)$ for $f'(x)$. Eqs. (3) and (4) are called the first-order approximation for $f'(x)$, as when $f'(x)$ is expressed, the lowest-order term has order of h with a coefficient proportional to $f''(x)$.

By adding Eq. (3) $\times \left(h_2^2\right)$ and Eq. (4) $\times \left(-h_1^2\right)$, we can cancel the $f''(x)$ term and obtain an approximation of $f'(x)$ with $O(h^2)$ in arbitrary h_1 and h_2:

$$f'(x) = \frac{h_1}{h_2(h_1 - h_2)} f(x+h_2) - \frac{h_1 + h_2}{h_1 h_2} f(x) - \frac{h_2}{h_1(h_1 - h_2)} f(x+h_1) + O(h^2) \qquad (5)$$

Group discussion 3.1:

> When $h_1 = -h_2 = h$, how can Eq. (5) be simplified? Observe further how the coefficients change when h_1 and h_2 are of opposite and same signs.

Equation (5) achieves second-order accurate approximation for the first derivative by cancelling out the term involving $f''(x)$ in the Taylor expansion of $f(x + h_1)$ and $f(x+h_2)$. In general, the Taylor series have the form:

$$\begin{aligned} f(x+h) &= f(x) + h \cdot f'(x) + \frac{h^2}{2!} f''(x) + \frac{h^3}{3!} f^{(3)}(x) + \ldots + \frac{h^n}{n!} f^{(n)}(x) + \ldots \\ &= \sum_{n=1}^{\infty} \frac{h^n}{n!} f^{(n)}(x) \cong \sum_{n=1}^{p} \frac{h^n}{n!} f^{(n)}(x) + O(h^{p+1}) \end{aligned} \qquad (6)$$

The last approximation in Eq. (6) is the "truncation", with an truncation error of order $p+1$ in the resoluation h. This is simple mathematically, but we can make several important observations:

- In addition to the sampling at x, if we know just one more point in the local analysis, the best we can do is $O(h)$. If we know two more points, we can have $O(h^2)$. Three, $O(h^3)$. This is just by cancelling out the terms involving h, h^2, h^3, and so on. The more sampling points we know around x, the higher-order approximation we can make for local analysis.

- Although when $h \to 0$, the high-order error terms diminish much faster, higher-order approximation may not be more accurate, because not only we do not know the pre-factors for the error terms, we do not know much about $f(x)$! For example, if $f(x)$ is an odd function, cancelling out terms of h^2 with $f''(x)$ does not make any improvement, as all even derivatives are zero there.
- When we approximate $f(x)$ with a Taylor expansion, the higher-order tems can cause local oscillation even with a slow-changing $f(x)$. Although when $h \to 0$, all oscillations have to diminish except in singular points as the higher-order terms have to become small (this is called "convergence" in the Taylor series, with the only exception that $f^{(n)}(x)$ diverges). When h is not so small limited by the finite precision in floating points, higher-order approximation is NOT equal to better or more accurate.
- As the approximation scheme is based on polynomial expansion in Taylor series, the approximation will fail to converge (or converge to the wrong thing) when we need to treat a function changing much faster than the polynomials in the local region, most commonly, the exponential or Gaussian functions. This is shown in the vicinity of 0 as:

$$\lim_{x\to 0} \frac{e^{-a/x}}{x^n} \to 0; \qquad \lim_{x\to 0} \frac{\exp\left(-\frac{a^2}{x^2}\right)}{x^n} \to 0 \qquad (7)$$

- Taylor series is not the only possible series expansion. Although Taylor series are more intuitive, the base functions of $1, x, x^2$, etc. are not orthogonal. Even within the polynomials, if we hope to obtain local analysis around $x = 0$, we can opt to use the orthogonal polynomial series such as Legendre series in Eq. (8) and Fig. 2. This can improve convergence and efficiency in determining the coefficients, as all terms are orthogonal.

$$L_n(x) = \frac{1}{2^n}\sum_{k=0}^{n}\binom{n}{k}^2 (x-1)^{n-k}(x+1)^k$$
$$L_0 = 1; \quad L_1 = x; \quad L_2 = \frac{1}{2}\left(3x^2-1\right); \quad L_3 = \frac{1}{2}\left(5x^3-3x\right)... \qquad (8)$$

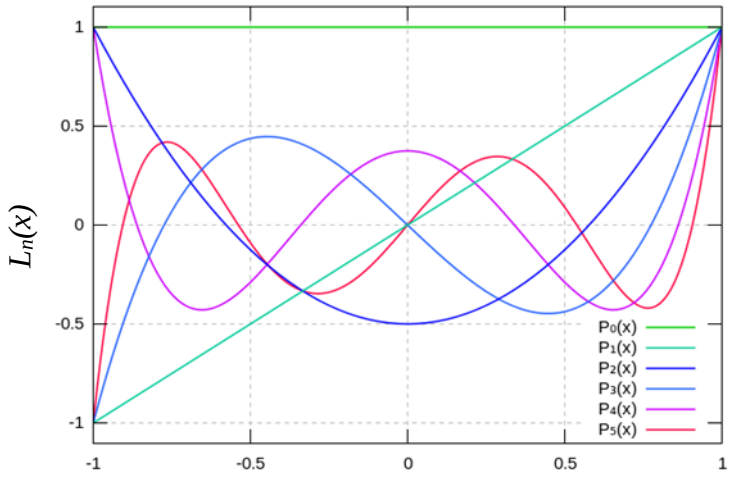

Fig. 2. Legendre polynomial functions in [−1, 1] for both x and y up to the 5th order.

- Some analtytical knowledge about *f(x)* in the vicinity of *x* will also surely improve choices of the more appropriate base functions, and hence better accuracy and adaptivity, as we can make predictions about the behavior of *f* by the choice of the base functions for approxmiation. This is especially useful for coupled equations (the variable x_1 depends on how another variable x_2 changes[1]), exponential functions (where Hermite series can be used), and known discontinuity (where we can have a discontinuity function as the base such as in discrete Galerkin in finite-element approximation).

3.2 Forward and backward Euler approximation

When the local approximation is with respect to time, an important rule will govern the stability of the ongoing projection by the present slope. We will treat this topic more thoroughly later in the ordinary differential equation, but I hope to leave a note here for you to know it may not be the same to make estimation from left, right or middle in the general case of h_1 and h_2 in Eq. (5). Consider the following dynamic equation of exponential growth ($a > 0$) or decay ($a < 0$) by an ordinary differential equation:

$$f'(t) = \frac{df(t)}{dt} = af(t) \qquad (9)$$

The solution is a form that you are familiar with: $f(t) = C\exp(at)$, where C will be given by the initial values of f at $t = 0$. For the first-order approximation with spacing of $\Delta t > 0$ when we have sampling at a previous time step $f(t - \Delta t)$ and the present $f(t)$, Eq. (9) can be evaluated by two ways:

$$\frac{f(t) - f(t-\Delta t)}{\Delta t} = af(t) \qquad \text{or} \qquad \frac{f(t) - f(t-\Delta t)}{\Delta t} = af(t-\Delta t) \quad (10)$$

The first expression estimates the right-hand side with the present time sampling. This is called the backward Euler method. "Backward" means from the $f(t)$ evaluation at the right-hand side, the slope is estimated in the past sampling. Simple algrebra will give:

$$f(t) = \frac{1}{1 - a\Delta t} f(t - \Delta t) \qquad (11)$$

When $a < 0$, $f(t)$ will have an exponential decay, and we can see that the prefactor $\frac{1}{1-a\Delta t}$ is always between (0, 1) regardless of the relation between a and Δt. The second expression in Eq. (10) estimates the right-hand side with the previous time sampling. This is called the forward Euler method:

$$f(t) = (1 + a\Delta t) f(t - \Delta t) \qquad (12)$$

We can see that $(1+a\Delta t)$ is between (0, 1) when $a < 0$ only for $\Delta t < -\frac{1}{a}$. $f(t)$ may be qualitatively wrong if we choose Δt inappropriately, unlike the situation in Eq. (11). Thus we call the backward Euler

[1] An example is for the electron concentration differentiation when the electric potential is solved by the Poisson equation. The electron concentration in the transport equation can be best approximated by the Bernoulli function of electrical potentials. Another example is the upwinding method in fluid dynamics, where the differentiation of driving force needs to be evaluated against the flow for stability.

to be unconditionally stable, as apart from accuracy, whatever time step is taken, an originally convergent progression will not blow up. This surely does not mean the inaccurate approximation we get has any use though. We will treat more stability issues in dynamic equations in later chapters. Here I just hope that you can see not all differentiation approximation with the same order will have similar behavior.

Hacker Practice 3.2:

For $f(t) = \exp(-t)$, i.e., $a = -1$, compare the evaluation of $f(t)$ for $0 \le t \le 20$ by three methods:

1. Ground truth: $f(t) = \exp(-t)$
2. Forward Euler with $f(0) = 1$ and march with $\Delta t = 0.5$, $\Delta t = 1.0$ and $\Delta t = 2.0$ by
   $f(t) = (1 - \Delta t) f(t - \Delta t)$
3. Backward Euler with $f(0) = 1$ and march with $\Delta t = 0.5$, $\Delta t = 1.0$ and $\Delta t = 2.0$ by
   $f(t) = \frac{1}{1 + \Delta t} f(t - \Delta t)$

Observe the "error" in Backward Euler in relation with Δt even with **absolute stability**.

3.3 Richardson extrapolation for error estimation and convergence analysis

A special case of Eq. (5) when $h_2 = 2h_1 = 2h$ deserves a closer look. The Taylor expansion looks like:

$$f'(x) = \frac{f(x+h) - f(x)}{h} + E(h); \qquad E(h) = O(h) = \frac{1}{2} h f''(x) + O(h^2) \quad (13)$$

$$f'(x) = \frac{f(x+2h) - f(x)}{2h} + E(2h); \qquad E(2h) = O(h) = \frac{1}{2} 2h f''(x) + O(h^2) \quad (14)$$

Here we use $E(h)$ and $E(2h)$ to represent the error in the expansion with $f(x + h)$ and $f(x + 2h)$, which contains the known h^2 term and everything else lumped into $O(h^3)$ in the expression. The second-order differentiation estimation from $f(x+2h)$, $f(x+h)$ and $f(x)$ now becomes:

$$f'(x) = \frac{-1}{2h} f(x+2h) - \frac{3}{2h} f(x) + \frac{2}{h} f(x+h) + O(h^2) \quad (15)$$

We can view Eq. (15) directly from Eq. (5), or they can be derived from Eqs. (13) and (14): a nested approximation of $f'(x)$ by $f(x + 2h)$ & $f(x)$, and then by $f(x + h)$ & $f(x)$, both of which originally are first-order approximation, but can be combined to become second order. We can use the same principle to add in the estimate from $f(x + 4h)$ & $f(x)$, or $f(x + 3h)$ & $f(x)$ to improve the approximation to the third order. The procedure can go on to even higher orders to formulate the p adaptivity. This nested procedure is called **Richardson extrapolation**.

Let's focus on the three sampling points of x, $x + h$ and $x + 2h$. Now we have three estimates of $f'(x)$ from Eqs. (13) – (15). Comparison of Eqs. (13) and (14) gives us an evaluation of the approximation improvement from finer gridding (h adaptivity), while comparison of Eqs. (13) and (15) an evaluation of the approximation improvement from raising the order (p adaptivity). From Hacker Practice 3.2, you should observe that h adaptivity works, but p adaptivity works far better as the function has important quadratic features. In Hacker Practice 3.1, when the function is indeed quadratic, there is NO error in the

approximation regardless of choice of *h* within the valid domain, i.e., the approximation is exact. In most real cases, we have limited knowledge of *f(x)* except the few sampling points (especially when *x* is a multi-variable vector, and evaluation of *f(x)* carries a significant computational or measurement cost), i.e., the ground truth is unknown or too expensive to be known. For example, $f(\varphi_1, \theta_1, \varphi_2, \theta_2, \ldots)$ is the distance (or vector) from the robotic palm tip to the object to be fetched, where (φ_i, θ_i) is the solid angle of the i-th robotic joints in a pseudo-rigid-body robotic arm. So, how can we verify if adaptivity by either *h* or *p* is effective?

We will look at the *h* adaptivity first. When error can be reasonably estimated, we know that the leading error in Eqs. (13) and (14) comes from the term associated with *f''(x)*:

$$R(h) \equiv \frac{E(2h)}{E(h)} \cong \eta \qquad (16)$$

R(h) is called the **Richardson extrapolation coefficient**. When *E(h)* is dominated by terms associated with *f''(x)*, $\eta \cong 2$. If we compute $\eta \cong 2$, it implies that errors from the higher order (h^2 and above in Eqs. (13) and (14)) are relatively small, and the local analysis will be in the "convergent domain" by the first-order approximation. Alternatively, when *E(h)* is dominated by the term associated with *f'''(x)* by using the second-order approximation such as Eq. (15), $\eta \cong 4$.

If error cannot be readily estimated due to lack of knowledge of the ground truth, we can perform the relative error estimate:

$$R(h) \cong \frac{\hat{A}(4h) - \hat{A}(2h)}{\hat{A}(2h) - \hat{A}(h)} \cong \eta \qquad (17)$$

where $\hat{A}(h)$ represents the local approximation function with sampling using *h*. For example, in the first-order approximation, $\hat{A}(h) = f'(x)\big|_h = \frac{f(x+h) - f(x)}{h}$. The approximation limit will apply. If the first order approximation is in the convergent domain, then η will be still close to $\frac{4-2}{2-1} = 2$. Notice that in Eq. (17), only the estimation $\hat{A}$ is needed, and the ground truth *A* is NEVER invoked.

We can apply the similar principle to compare the *p* adaptivity, which we will do in the later treatment of ordinary differential equation (ODE) when we can give more realistic examples.

One more interesting observation can be made. We can use the *p* error estimate to guide the choice of *h*! This is a VERY important technique, as choice of *h* will not only affect errors as shown in Fig. 1, but is the dominant factor for the computational cost. For example, *h* is the grid resolution in a 3D geometry. The number of grid points will be proportional to $1/h^3$ (homogeneous gridding). If we assign a variable to every grid point (such as temperature) and try to solve a thermal conduction equation (an elliptic Laplace equation), whose computational cost is often proportional to n^2 or n^3 with *n* being the number of variables. Now our computational cost is proportional to $1/h^6$ or even $1/h^9$!!! Refining *h* to *h/2* in gridding of this naïve example will cause $2^6 = 64$ to $2^9 = 512$ times of computational cost!!! This makes the smart choice of *h* absolutely critical!!!!

Hacker Practice 3.3:

For $f(x) = x^3$, we know the exact $f'(x=1) = 3$. Use Eqs. (13) – (15) to estimate $f'(x=1)$ varying the value of h from 2^{-4} to 2^{-20} to observe the relative error in calculating $f'(x)$. Estimate η from Eqs. (16) and (17) for each choice of h.

4 Integration and interpolation

Now we will look at the integration over a segment, which mathematically is also called the "quadrature[2] scheme".

$$I_k = \int_{x_k}^{x_{k+1}} f(x)dx; \qquad h_k = x_{k+1} - x_k \tag{18}$$

Integration is important to look at the "average" behavior or find the general trends, and is also extenisvely used in the finite-element method to find how the approximation function approaches the real function within the "element". Let's assume that discretization has already been performed for the region of interests, i.e., in 1D, the region of interests of (a, b) has already been diced into N segments of h_k (k = 1, … N). Error estimation and adaptation by geometrical resolution (h) can be performed similarly to differentiation, as the finer h resolution will give a better approximation to the analytical integration in Eq. (18). We will now only focus on the approximation function for integration and its associated p adaption.

We will first perform axis shift and normalization for $f(x)$, so that $x_k = -1$, $x_{k+1} = 1$, as shown in Fig. 3. This may give additional pre-factors to I_k, but will not affect the general conclusion. This normalization will make the integration approximation clearer, especially for the Taylor and Legendre series. A general method will be using summation to transform an integration to a weight sum as:

$$\int_a^b f(x)dx \cong \sum_{k=1}^{N} h_k \left(w_k f(x_k)\right) = \sum_{k=1}^{N} \hat{I}_k = \hat{I} \tag{19}$$

where w_k is a weighting number to consider how $f(x_k)$ should be counted towards the contribution of evaluating the average of $f(x)$ in the interval of h_k. The first effect we can observe for this normalization within ($x_k = -1$, $x_{k+1} = 1$) is that all odd-functions (i.e., $f(x) = -f(-x)$) will not give any contribution to the integral. For polynomials, only constant, x^2, x^4, etc. will give contribution to the quadrature. Therefore, we will start from second-order accuracy if we sample (−1, 1) evenly. Higher orders of approximation will be 4th, 6th, and so on.

(**Group Discussion**) By employing the even and odd function properties after shifting and normalizing the x period, we achieve better order of accuracy for integration estimation. Is there a penalty from the point of view in interpolation?

Table 1 summarizes the common integration approximation. Notice that the rectangle method samples only on $f(-1)$ asymmetrically in the interval of (−1, 1), and is therefore only first order. Any symmetric

[2] Numerical quadrature means finding the value of a definite integral, or equivalently, finding the solution of the associated differential equation. This can be for 1D or higher dimension.

sampling like trapezoid, midpoint, Simpson and Gaussians, will eliminate the contribution from the odd function (x, x^3, etc.) and hence be even-order of accuracy.

Table 1. Normalized qudrature approximation for integration between (–1, 1)

Type	Approximation	Order of precision
Rectangle	$\hat{I}_k = h_k f(x_k)$	1st order
Trapezoid	$\hat{I}_k = h_k\left(\frac{1}{2} f(x_k) + \frac{1}{2} f(x_{k+1})\right)$	2nd order
Midpoint	$\hat{I}_k = h_k f(x_{k+1/2})$	2nd order
Simpson	$\hat{I}_k = h_k\left(\frac{1}{6} f(x_k) + \frac{4}{6} f(x_{k+1/2}) + \frac{1}{6} f(x_{k+1})\right)$	4th order
2-point Gaussian	$\hat{I}_k = h_k\left(\frac{1}{2} f\left(x_{k+1/2} - \frac{1}{2\sqrt{3}} h_k\right) + \frac{1}{2} f\left(x_{k+1/2} + \frac{1}{2\sqrt{3}} h_k\right)\right)$	4th order
3-point Gaussian	$\hat{I}_k = h_k\left(\frac{5}{18} f\left(x_{k+1/2} - \frac{\sqrt{3}}{2\sqrt{5}} h_k\right) + \frac{8}{18} f(x_{k+1/2}) + \frac{5}{18} f\left(x_{k+1/2} + \frac{\sqrt{3}}{2\sqrt{5}} h_k\right)\right)$	6th order

We will use an example in Fig. 3 to give you some quantitative sense of the quadrature scheme. For $f(x) = 7x^3 - 8x^2 - 3x + 3$, with the integral between (–1, 1) being equal to 2/3. The rectangle approximation will give $-9\times2 = -18$ (very bad). The trapezoidal rule will give $0.5\times(-9 + (-1))\times2 = -10$. The midpoint will give $3\times2 = 6$. We can see that all these 1st and 2nd order approximation can be pretty bad as it does not capture the detailed variation. The Simpson rule, which needs 3 sampling point, will be

$$\hat{I}_k = h_k\left(\frac{1}{6} f(x_k) + \frac{4}{6} f(x_{k+1/2}) + \frac{1}{6} f(x_{k+1})\right) = 2\times\left(\frac{1}{6}\cdot(-9) + \frac{4}{6}\cdot 3 + \frac{1}{6}(-1)\right) = \frac{2}{3}$$

This 4th order scheme captures the detailed variation and already achieve the exact answer!

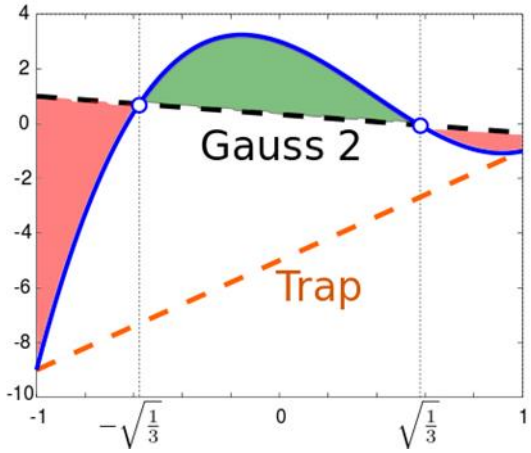

Fig. 3. Integration approximation is shown on a normalized segment of (-1, 1). The function $f(x)$ in the blue line to be integrated here is: $f(x) = 7x^3 - 8x^2 - 3x + 3$, with the integral equal to 2/3. Comparison between 2-point Gaussian (black dash line) and trapezoidal (orange dash line) quadrature is shown. The trapezoidal rule returns the integral of -10, although it is a second-order scheme. The 2-point Gaussian quadrature rule returns the integral of 2/3 exactly as a 4th-order scheme, which will return accurate results for any function that can be well approximated by a cubic polynomial.

Presume we are looking at the normalizd and shifted $f(x)$ is polynomial as $f(x) = a_0 + a_1x + a_2x^2 + a_3x^3 + a_4x^4 + \ldots$ We first notice that in all quadrature approximations in Table 1, in order to evaluate the integration of constant a_0 correctly (as $2a_0$ after integration between (−1, 1)),

$$\sum_{i=1}^{N} w_i = 1 \qquad (20)$$

4.1 Gaussian quadrature schemes

We can now focus on how to integrate a_2x^2 correctly to achieve 4th order accuracy (as x and x^3 terms will not contribute to the integrand). The analytical solution here is

$$\frac{a_2}{3}\left(x_{k+1}^3 - x_k^3\right) = \frac{a_2}{3}\left(\left(x_k + h_k\right)^3 - x_k^3\right) = \frac{a_2}{3}\left(3x_k^2 h_k + 3x_k h_k^2 + h_k^3\right) \qquad (21)$$

For the Simpson rule with sampling at the boundary and the midpoint, the Taylor expansion gives:

$$\begin{aligned}
&h_k\left(\frac{1}{6} f\left(x_k\right) + \frac{4}{6} f\left(x_{k+1/2}\right) + \frac{1}{6} f\left(x_{k+1}\right)\right) \\
&= h_k\left(\frac{1}{6} f\left(x_k\right) + \frac{4}{6}\left(f\left(x_k\right) + \frac{h_k}{2} f'\left(x_k\right) + \frac{h_k^2}{8} f''\left(x_k\right)\right) + \frac{1}{6}\left(f\left(x_k\right) + h_k f'\left(x_k\right) + \frac{h_k^2}{2} f''\left(x_k\right)\right)\right) \\
&= h_k\left(f\left(x_k\right) + \frac{h_k}{2} f'\left(x_k\right) + \frac{h_k^2}{6} f''\left(x_k\right) + O\left(h_k^3\right)\right) = h_k\left(a_2 x_k^2 + \frac{h_k}{2}\left(2a_2 x_k\right) + \frac{h_k^2}{6}\left(2a_2\right) + O\left(h_k^3\right)\right) \\
&= \frac{a_2}{3}\left(3x_k^2 h_k + 3x_k h_k^2 + h_k^3\right) + O\left(h_k^4\right) \qquad for \quad f\left(x\right) = a_2 x^2
\end{aligned} \qquad (22)$$

where we can find that the approximation is exact for a_0 and a_2 terms in $f(x)$.

This is the Taylor expansion that comes up with the Simpson rule to approximate the integral of $a_0 + a_2x^2$ exactly. The great mathematician Carl Gauss found that sampling at 2 points, not 3, can achieve the same approximation. This can be observed by the following equality from Eq. (19):

$$\begin{aligned}
&w_1 f\left(x_k + \ell_1\right) + w_2 f\left(x_k + \ell_2\right) = f\left(x_k\right) + \frac{h_k}{2} f'\left(x_k\right) + \frac{h_k^2}{6} f''\left(x_k\right) \\
&\quad = w_1\left(f\left(x_k\right) + \ell_1 f'\left(x_k\right) + \frac{\ell_1^2}{2} f''\left(x_k\right)\right) + w_2\left(f\left(x_k\right) + \ell_2 f'\left(x_k\right) + \frac{\ell_2^2}{2} f''\left(x_k\right)\right)
\end{aligned} \qquad (23)$$

Matching the coefficients of $f(x_k)$, $f'(x_k)$ and $f''(x_k)$, we obtain:

$$w_1 + w_2 = 1$$

$$w_1\ell_1 + w_2\ell_2 = \frac{h_k}{2} \qquad (24)$$

$$w_1\ell_1^2 + w_2\ell_2^2 = \frac{h_k^2}{3}$$

We have four variables and 3 constrained equations, which means infinite choices can satisfy the requirement of 4th-order quadrature approximation with two sampling points. If we choose w1 = w2 = ½, then we have the choices of the conventional 2-point Gaussian quadrature:

$$\ell_1, \ell_2 = \frac{1}{2}\left(1 \pm \frac{1}{\sqrt{3}}\right) \qquad (25)$$

We can check by direct substitution to Table 1 as well:

$$\begin{aligned}
&h_k\left(\frac{1}{2}f\left(x_{k+1/2} - \frac{1}{2\sqrt{3}}h_k\right) + \frac{1}{2}f\left(x_{k+1/2} + \frac{1}{2\sqrt{3}}h_k\right)\right) \\
&= h_k\left(\begin{array}{l}
\frac{1}{2}\left(f(x_k) + \left(\frac{h_k}{2} - \frac{h_k}{2\sqrt{3}}\right)f'(x_k) + \frac{1}{2}\left(\frac{h_k}{2} - \frac{h_k}{2\sqrt{3}}\right)^2 f''(x_k)\right) \\
+\frac{1}{2}\left(f(x_k) + \left(\frac{h_k}{2} + \frac{h_k}{2\sqrt{3}}\right)f'(x_k) + \frac{1}{2}\left(\frac{h_k}{2} + \frac{h_k}{2\sqrt{3}}\right)^2 f''(x_k)\right)
\end{array}\right) \qquad (26) \\
&= h_k\left(f(x_k) + \frac{h_k}{2}f'(x_k) + \frac{h_k^2}{6}f''(x_k) + O\left(h_k^3\right)\right) \\
&= \frac{a_2}{3}\left(3x_k^2 h_k + 3x_k h_k^2 + h_k^3\right) + O\left(h_k^4\right) \quad for \quad f(x) = a_2 x^2
\end{aligned}$$

Similar quadrature schemes can be extended to arbitrary orders of precision for minimal sampling points.

We can see how hp adaptivity can be applied here, similar to the Richardson extrapolation in the local analysis of differentiation. Similar to hp adaptivity in the local analysis for differentiation:

1. Making *h* smaller, we can have better integration approximation according to O(h^p).
2. Making the order of approximation higher (*p* adaptivity), we can have better integration approximation on the segment *h*.
3. We can use different choices of h_k to compare the relative accuracy of approximation.
4. We can use different choices of order of approximation to compare the relative accuracy of approximation, and can then choose the appropriate h_k!

Hacker Practice 3.4:

For $f(x) = e^x$, we know the exact solution to the integration as:

$$\int_{-1}^{1} e^x dx = e - e^{-1} \cong 2.3504$$

Notice that this is a monotonic function, so the order of approximation will not cause big deviation as the cubic function before. Implement the rectangle, trapezoid, mid-point, Simpson and 2-point Gaussian approximation and tabulate your results.

5. Modular objects and unit testing

5.1 Scope with least coupling in the procedural interface

We have learned how to construct a class in C++ to achieve data hiding by pinning down the "method" to access the data structure instead of the data structure itself. We have claimed that this is very important for modular programming. Now we need to further ask the question: how do we choose a class or object? How can the object, as well as all of its associated functions and procedural calls, have a smallest possible "scope" to achieve modular programming?

The answer is for sure problem dependent, but similar to the problems in organization behavior, millitary command lines, and even placement and routing in integrated circuits. Given a problem, you have tried out the first procedural interface design to link each object as shown in Fig. 4. The objects of your first designs may be of a large variation in number of functional calls and in interaction with other objects. Modular programming is achieved in an iterative process:

1. We will need to first identify utilities objects, similar to janitoring, plumbing and electrician support in an organization. They are coupled to most objects, but a clean interface can be relatively easy to define. These objects should become a library or a utility and remove from the object map. Examples are like vector manipulation, matrix operations, stack handling, hash table, etc.
2. For each object, if other objects will need to use one of its procedural calls, draw a line with arrows to indicate data flow. The size of the block to represent the object is proportional to its number of the procedural interface calls.
3. Set a goal for the number of top modules you would like to have. This is also related to software management and programmer delegation, in addition to the logic breakdown of your codes. Ideally we should have objects of reasonable size distribution. Smaller objects (not utilities) should be considered for merging and larger objects should be considered for splitting or putting into a class hierarchy.
4. Find cut lines that can go through the least number of interconnecting lines but enclose the most number of objects. Each enclosure suggests a possible good topological division for the whole codes. The enclosed object can be considered for becoming a new larger object or a superclass of a group of objects. This is similar to placement and routing in integrated circuits design.
5. Go through iterations of object designs and present in a software architecture review. Refine the modules to achieve clean module separation and clear delegation of software management.
6. Attempt defining testing methods within the objects (test main( ) goes through each procedural interface to validate implementation) and across the interacting objects (test main( ) imitate calls

of objects that interact with other objects). A good modular design should include a good, well-balanced design of unit testing. If tests cannot be easily defined, it is likely your modular design needs improvement.

An example of a general-purpose, dial-an-operator (meaning that users can dial in any physical operators such as Laplacian, differentiation, nonlinear functions, etc.) finite-element solver is shown in Fig. 5.[3] The tool originally developed for CAD platform had evolved for several generations and adopted by industry such as Intel, IBM and Bell Labs, until COMSOL by MathWorks came in 2005 to dominate that application sector.

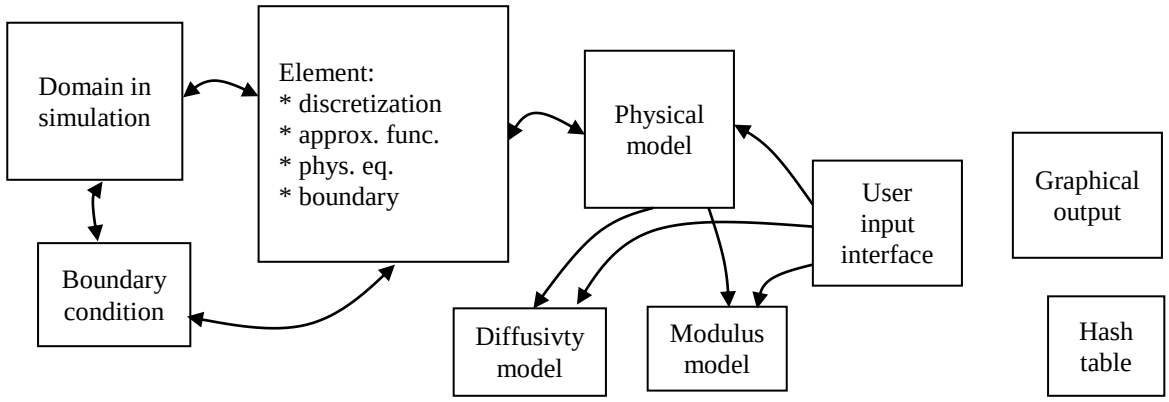

Fig. 4. Object models for modular programming planning. The objects of hash table and graphical output are recognized as utilities, and the element object is considered too large.

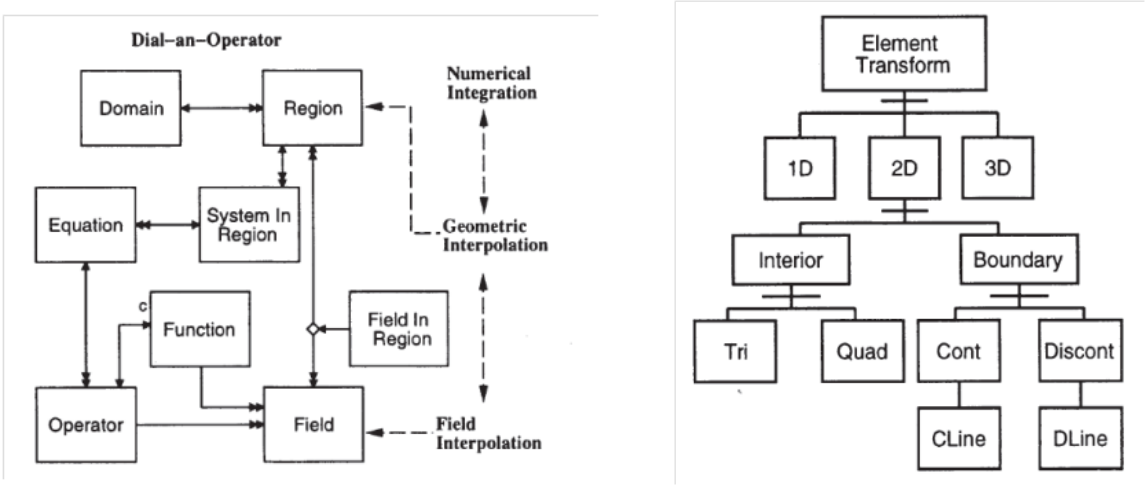

Fig. 5. Examples of object maps and superclass inheritance structures in a general-purpose, dial-an-operator finite-element solver. The single arrow indicates read operations and double arrows indicates read/write operations. The small diamond represents a specific association relationship.

5.2 Unit testing within the scope and cross testing

[3] D. W. Yergeau, E. C. Kan, M. Gander and R. W. Dutton, "ALAMODE: a layered model development environment," *Simulation of Semiconductor Devices and Processes*, H. Ryssel and P. Pichler, eds., 1995, p. 66.

Through the object definition process (with iterations), we can then try to design for unit testing within each object scope and for each object link. Remember that this will involve the object itself (your test program calls the object method) and any other object that has an interaction with this object (your test program will call the object that interacts with the object in scope). A few principles can be observed:

1. Within each object scope, unit testing should be well confined, if not, it probably indicates that further object design iterations are needed.
2. A good object map will give the smallest possible number of needed tests to comprehensively cover the whole map.
3. The unit and cross testing should be, and is clearest to be, designed together with the object map.
4. The object map together with the testing suites is an important tool in software development management and reporting.

Modular programming of source codes and unit testing should be under the same version control as the center of the code evolution. Without doing so, when a bug needs to be fixed or a new function needs to be implemented, we will be dealing with a jungle!

6. Software Engineering: Source Code Control with Regressional Testing

6.1 Source code control basics

Consider a software development team with many programmers in a hierachical structure. For example, in Microsoft, (fictitious estimate) 2,000 programmers are assigned for Office Word development. 1,000 programmers may be assigned to develop test codes and suites, and 500 programmers for the logic flow and Word-internal implementation, and 500 programmers for interfacing with Windows and iOS for input commands and screen rendering. Within the 1,000 testing programmers, say 300 are assigned to development code testing and 700 for post-development specification testing. All these programmers need to work on the SAME repository of codes that can be managed to produce reliable product. Source code control and management are absolutely necessary for such a large and coordinated efforts, with codes evolving with the testing suites of units, modules, domains and specification. From the early days of software development, code management tools are written together with the operating system itself. This is good as the OS and its applications will need source code management, but is somewhat inconvenient as code management should not be limited to one OS.

The minimal feautures in a typical source code control system include:

- A central repository where all versions of various files have a time stamp, a modification history (time and author) and the association with other files (this is sometime maintained by the Unix "makefile").
- Users can link to that repository to **checkout** files, create **branches** and **commit** back modification with a signer's approval. The users should have his own working directory independent of the repository.
- Different versions from difference in time, branch or authors can be compared to generate a list of differences for **merge**.
- A **signer** or captain or head or Lord (many names describing this "boss" role) is in charge of the repository for adding new files, delete files and dead branches, merge approval, conflict resolution and system backup.

For a long time, this software management is always done on top of the Unix/Linux file system. We have three classic tools for software management:

- SCCS: Software Configuration and Change Management
- CVS: Concurrent Versions System
- SVN: Apache Subversion (popular all the way to around 2007)

As SVN/CVS/SCCS are based on the Unix file system, the commands are mixed with Unix file directory supports. Hierarchical controls are built into the hierarchical file system, and a **central repository** is assumed. In the early days, the memory is scarce, and the default is to store the difference (delta) between different versions, instead of snapshot. Snapshots can be forced by explicit commands from the user or the repository manager.

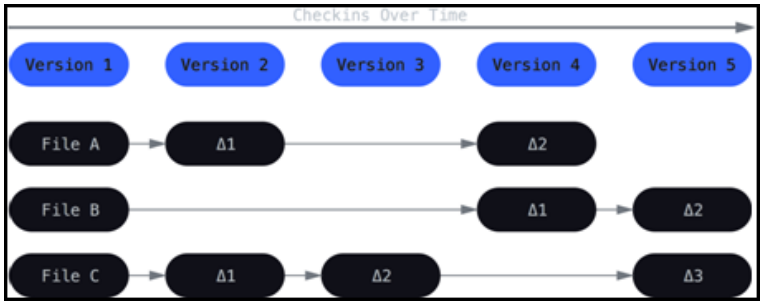

Fig. 6. SCCS/CVS/SVN have default storage between versions as delta.

6.2 The new standard: Git

From 2007, a new trend in software management has been pioneered by Github, a new company then headed by Chris Wanstrath. The product, offered most often freely to everyone with Internet access, is called Git. Github is evaluated today at more than $2B[4] and its main business focuses on software, document and project management. Although Git and SVN/CVS/SCCS have many similar feature and philosophy, their main difference include:

1. Git does not assume a default file system, and can work seamlessly across various OS such as Linux, Windows, iOS and Android.

2. Git always stores snapshots because source codes and test suites are mostly text files which are small in today's storage system, as shown in Fig. 7.

3. Git implements a hierarchical "**local**" and "**remote**", instead of assuming a central repository like SVN/CVS/SCCS. People can freely create their own node as local, and symbolically linked to a remote under the permission of that remote. A tree of local and remote modes can be built, similar to military or company hierarchy.

[4] https://en.wikipedia.org/wiki/GitHub

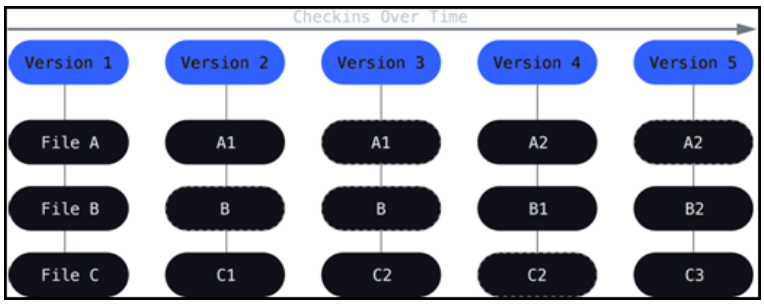

Fig. 7. Storing snapshots of different versions always, as in Git.

The third point is most powerful, which makes Git a new standard for not only code development but also any documents. A local node structure is illustrated in Fig. 8. A "signer", "head" or "captain" will be assigned repository management right to sign for merge, commit and connection to other "locals" as their "remote". Notice that this "head" cannot write into the private area of his "remote" or invade the working directory of his developers, but can only send requests.

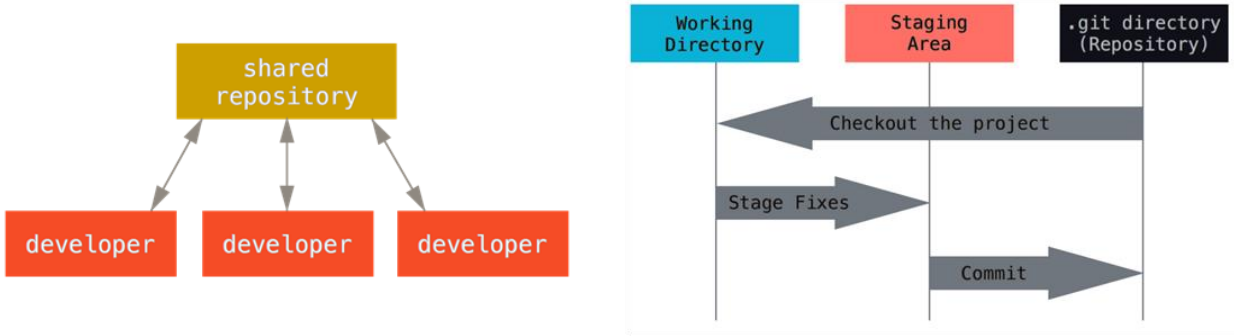

Fig. 8. The local node contains a local repository for coordination of multiple developers in Git. The working directory (belonging to each developer with read/write permission) interacts with the repository (belonging to the "signer" with maintenance rights) through a "staging area" (comments, requests and approval).

The staging area in a local node serves as the buffer zone between the local developers and head. Each developer has his/her own working directory that the head does not need to have access, and the head is in charge of the repository management to resolve conflicts among developers during different merge, to approve commit for a new version, and to add/delete the repository file structure. Alternatively, as shown in Fig. 9, each developer can create private and public areas to interact with the head/manager, or a local hierarchy can be built for one "dictator" (the real head for the local repository) who is in charge of several lieutenants, who are then working with their groups of developers. To enable concurrent code development from various developers, a local branch and merge in Fig. 10 can be done with agreement of branch owners or the "head".

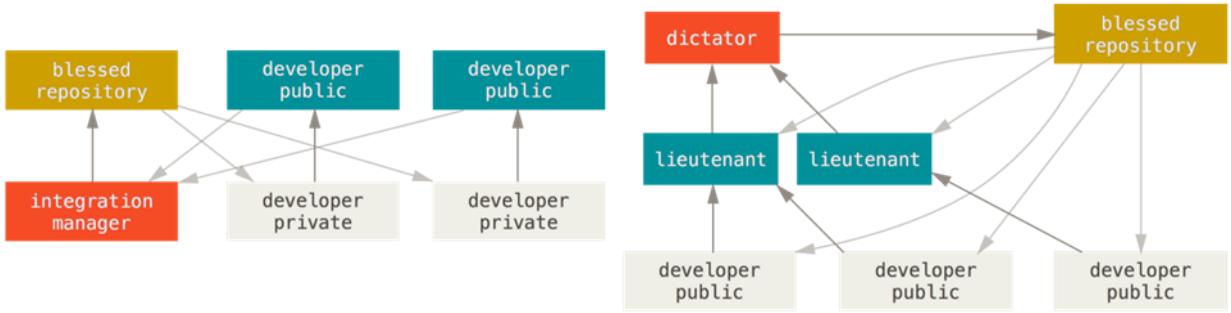

Fig. 9. Alternative structures for a local node.

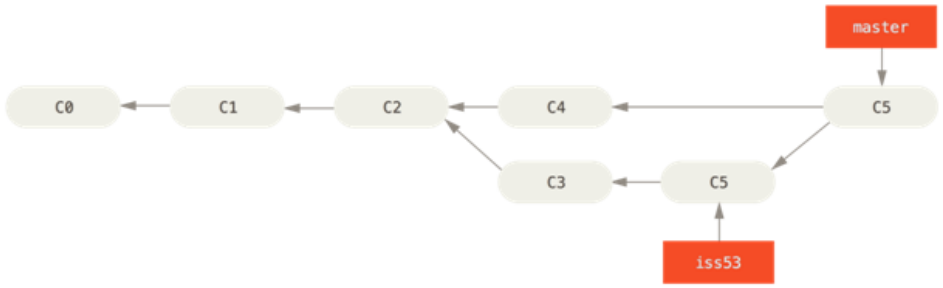

Fig. 10. An example of local branch and merge to allow concurrent development.

Similar master-slave relations are between a local "head" and the head of his "remote". The major commands of Git, in comparison with CVS as a historical example, are listed in Table 3. Notice that Git inherits a lot of CVS commands.

Table 3. Comparison between basic Git and CVS commands.

Git Commands	CVS compatibles	Comments
`git config`	`(UNIX $whoami)`	Author registration
`git init`	`(UNIX $mkdir cvs)`	Create a local repository
`git clone`	`(UNIX $cp cvs user/cvs)`	Create a working copy from local or remote
`git add`	`cvs add`	Add one or more files to stagging; Perform manual merge after resolving conflicts
`git commit`	`cvs commit`	Check files into local head repository
`git push`	`N/A`	Check changes into master or remote repository
`git status`	`cvs status; admin; history`	List files that you add, and need to add or commit
`git remote`	`N/A`	Link local repository to remote master
`git checkout`	`cvs checkout`	Create a new branch
`git branch`	`cvs history; log; delete`	List all branches in history; Manage/delete branches
`git push`	`N/A`	Push your repository to public remote; Manage your own push (delete, notify, etc.)
`git pull`	`N/A`	Fetch and merge changes from remote to local
`git merge`	`(cvs release)`	To merge different branches into one branch
`git diff`	`cvs diff`	View merge conflicts; Difference between branches
`git tag`	`cvs tag`	Adding attributes to any file or branch
`git fetch; reset`	`?`	Drop all local files and refresh from remote
`git grep`	`(UNIX $grep)`	Search strings in all local or remote repository

Git has additional commands to handle the relation between "local" and "remote", which does not exist in SVN/CVS where a central repository is assumed.

6.3 Software code and test suite maintenance

Together with the source code control, the testing and backup are integrated into the scheduled maintenance as listed below.

The daily maintenance:

- A small test suite is run on all branches with the committed version. The daily test suite is designed to finish in 6 – 8 hours with validation and execution time reports. A bug report with execution time is logged.
- A differential or snapshot system backup is performed. The new system copy from last-week's frozen version will be stored with at least two medias (two hard drives, or one local drive and one remote drive, or some companies still maintain DVD or tapes).

The weekly maintenance:

- A fuller test suite is run on all branches with the committed version. The weekly test suite is designed to finish in 40 – 60 hours with validation, execution time profiles and memory usage reports.
- A semi-stable version is often forced and named as: yyww, where yy is the year and ww (01 – 52) is the number of week in a year.

The major version maintenance:

- All source code and test suites are frozen as a development tree in CVS or Git repository. This record is not only for programmers to understand the flow of design and logistics, but also needed in case of legal dispute of copyright and intellectual properties.
- Specification and user manual have to be consistent with the present version.
- A full bug report of history, validation, performance profiles and memory usage is maintained.