# LEARNING DIARY

## Numerical Analysis

NGUYEN XUAN BINH 887799
MAJOR: COMPUTATIONAL ENGINEERING

# Table Of Contents

# Round 1: Solution of non-linear equations

## 1. Comments on the lecture and exercise materials

Regarding the floating point explanation, I think the lecture notes could be better if it explains in depth how the exponent and the mantissa are calculated.
The order of convergence formula is not given in the lecture notes as well, which is

$$\alpha = \frac{\ln\left(\frac{e_{n+1}}{e_n}\right)}{\ln\left(\frac{e_n}{e_{n-1}}\right)}$$

Where e_n is the error at iteration n. Another formula for the order of convergence not given in the lecture notes but it is used in exercise 1 is this formula:

Suppose $\{x_k\}_{k=0}^{\infty}$ is a sequence that converges to x*, with $x_k \neq x^*$ for all n. If positive constants $\lambda$ and $\alpha$ exist with

$$\lim_{k\to\infty} \frac{|x_{k+1} - x^*|}{|x_k - x^*|^{\alpha}} = \lambda$$

Then $\{x_k\}_{k=0}^{\infty}$ converges to p of order $\alpha$, with asymptotic error constant $\lambda$

In Exercise 2, the lecturer should also designate the range of floating point numbers that for n >= 53, 2^(-n) cannot be recognized for floating point numbers.

## 2. Reflections on what I have learned

In this round's materials, I found the fixed point method to be quite strange but interesting, since using the output as an input for the function turns out to convergent to the root of the equation. Proving the convergence of the fixed point method is quite of a challenge for me. What I can only try is just to plug in the point and let it recurse for many times to see if the value converges or diverges. I haven't found out a solid solution to prove any given fixed point method to be convergent.
Also, the Steffensen's method is new to me because I have not seen it before. It could be better if the lecturer can cover this method in the lecture notes as well.

## 3. Newton's method

In mathematical optimization, the Newton's method can be used to find the local optima of a function within a range. A level of tolerance is usually set beforehand [1]

**Algorithm** Newton's method

1: **initialise.** tolerance $\epsilon > 0$, initial step size $x_0$, iteration count $k = 0$.
2: **while** $|f'(x_k)| > \epsilon$ **do**
3: $\quad x_{k+1} = x_k - \frac{f'(x_k)}{f''(x_k)}$.
4: $\quad k = k + 1$.
5: **end while**
6: **return** $\overline{x} = x_k$.

Newton's method is the main method employed in many optimisation algorithms. However, it has convergence issues if x0 is too far away from optimal. In other words, Newton method is not guaranteed to always converge. For quadratic problems, the approximation for the optima is exact, meaning that only one iteration is needed [1]
This Newton's method is a modified version of the original formula, in that it is used to find the root of the derivative of a given function. Therefore, the original Newton's method is used to find the roots, while this version is used to find the inflectional points.

## 4. Bisection method

Similarly like the Newton's method, the bisection method can be used to find the local optima of a function within a range. However, the second derivative of the function is not needed like the Newton's method [1]

**Algorithm** Bisection method (minimisation)

**initialise.** tolerance $l > 0$, $[a_0, b_0] = [a, b]$, $k = 0$.
**while** $b_k - a_k > l$ **do**
    $\lambda_k = \frac{(b_k + a_k)}{2}$ and evaluate $f'(\lambda_k)$.
    **if** $f'(\lambda_k) = 0$ **then** return $\lambda_k$.
    **else if** $f'(\lambda_k) > 0$ **then**
        $a_{k+1} = a_k$, $b_{k+1} = \lambda_k$.
    **else**
        $a_{k+1} = \lambda_k$, $b_{k+1} = b_k$.
    **end if**
    $k = k + 1$.
**end while**
**return** $\overline{\lambda} = \frac{a_k + b_k}{2}$.

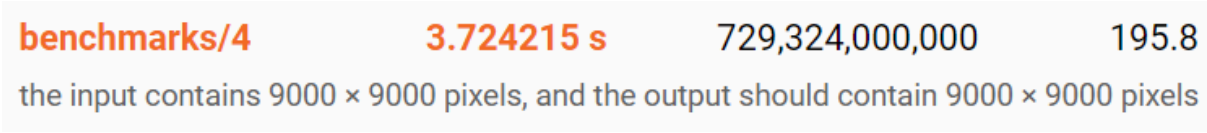
For example, we can calculate the numerical result of the optima without resorting to analytical derivation. In the function -2log(x) + (x-1)^2, its global minimum is 1.618

```
a = 1 # starting interval
b = 2 # starting interval
ϵ = 0.001 # precision

#function we want to optimise
f(x) =  -2*log(x) + (x-1)^2

# call bisection function
min_bisection(f,a,b,ϵ)
```

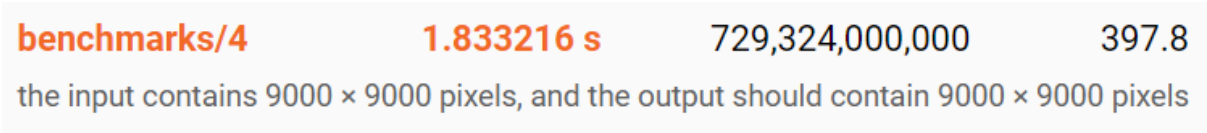
```
a_0 = 1.5 b_0 = 2
a_1 = 1.5 b_1 = 1.75
a_2 = 1.5 b_2 = 1.625
a_3 = 1.5625 b_3 = 1.625
a_4 = 1.59375 b_4 = 1.625
a_5 = 1.609375 b_5 = 1.625
a_6 = 1.6171875 b_6 = 1.625
a_7 = 1.6171875 b_7 = 1.62109375
a_8 = 1.6171875 b_8 = 1.619140625
a_9 = 1.6171875 b_9 = 1.6181640625
λ = 1.6181640625
```

### 5. Floating-Point Arithmetic

In parallel computing, arithmetic done with floats is about twice faster than doubles, since double is twice the precision of floats [2]. This is illustrated below, where 729 billion arithmetics are calculated by a multicore computer.




The double arithmetic takes 3.72 seconds




while float arithmetic takes half the time, which is 1.83 seconds

Conversions from double to float will also cause rounding errors if multiple additions of doubles are accumulated over a sum of type float [2]

Another inherent problem present in floating point number arithmetic is that it does not result in exact solution. For example, in Scala compiler, the addition of 0.1 + 0.2 is not 0.3, but it is actually 0.1 + 0.2 = 0.30000000000000004. I used to have coded the banking API transactions before and it is suggested that we should not use floating point numbers. Instead, the Python package library Decimal is helpful to calculate the exact values.

## Round 2: Polynomial interpolation

### 1. Comments on the lecture and exercise materials

Considering the barycentric formula of the Lagrange interpolation, the addition of one more interpolation with the constant function 1 besides the original data should be covered in the exercise hints materials. I found lacking this crucial idea will be troublesome for students to find the correct direction

Additionally, Newton's interpolation should be added in the exercise set as well, because after the course I haven't had a chance to know how Newton's interpolation looks like compared to Lagrange method.

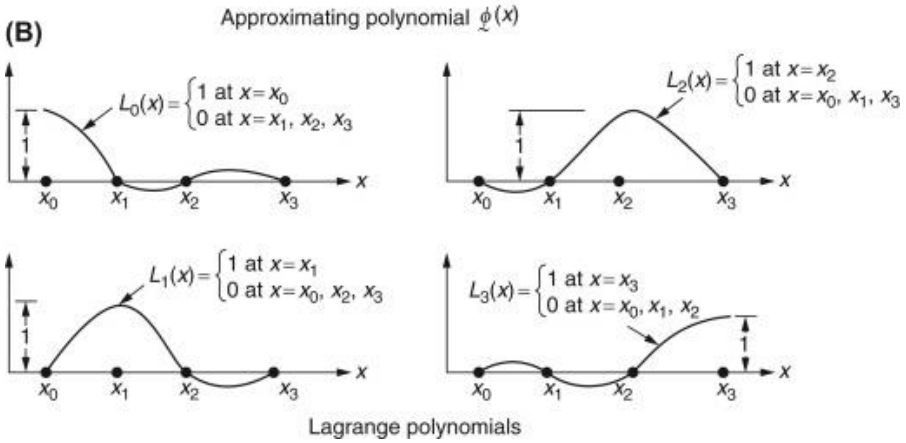
### 2. Reflections on what I have learned

The Lagrange interpolation appears to be a useful method for interpolation inside the data points. I especially like the barycentric formula, which can be updated efficiently in O(1) time when new data points are added.

Furthermore, I have a chance to perform operations on sumations of many variables in the the proof, which helps me consolidate my understanding of summation arithmetics

### 3. Finite Element Method

Lagrange interpolation is a well-known method to in predicting values inside a mesh in the finite element method. FEM uses discretization of the material surface to predicting different physical values on the object based on initial boundary conditions.




Source: Xinwei Wang, in Differential Quadrature and Differential Quadrature Based Element Methods, 2015 at https://www.sciencedirect.com/topics/engineering/lagrange-interpolation-function

By using the lagrange interpolation, we can calculate the values of the nodes inside the mesh. Another possible candidate function for FEM is in the form [3]

$$\frac{k}{\Delta x}(a_{i-1} - 2a_i + a_{i+1}) + F_i + f'\Delta x = m'\frac{\Delta x}{6}(\ddot{a}_{i-1} + 4\ddot{a}_i + \ddot{a}_{i+1}) \qquad i \in \{1, 2, \ldots, n-1\}$$

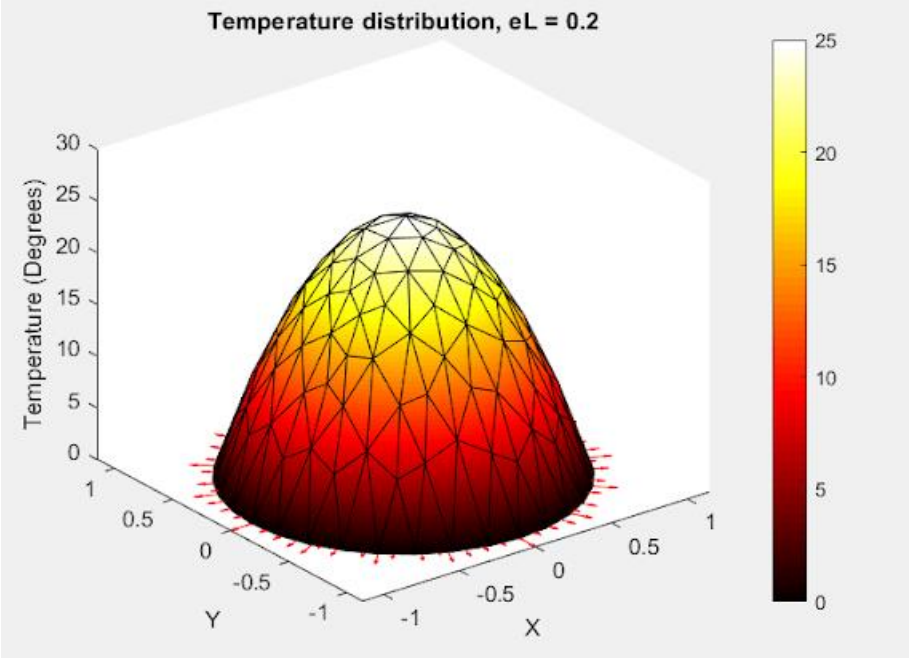
$$\frac{k}{\Delta x}(a_1 - a_0) + F_0 + f'\frac{\Delta x}{2} - m'\frac{\Delta x}{6}(2\ddot{a}_0 + \ddot{a}_1) = 0 \quad \text{or} \quad a_0 = \underline{a}_0,$$

$$\frac{k}{\Delta x}(a_{n-1} - a_n) + F_n + f'\frac{\Delta x}{2} - m'\frac{\Delta x}{6}(2\ddot{a}_n + \ddot{a}_{n-1}) = 0 \quad \text{or} \quad a_n = \underline{a}_n,$$

$$a_i - g_i = 0 \quad \text{and} \quad \dot{a}_i - h_i = 0.$$

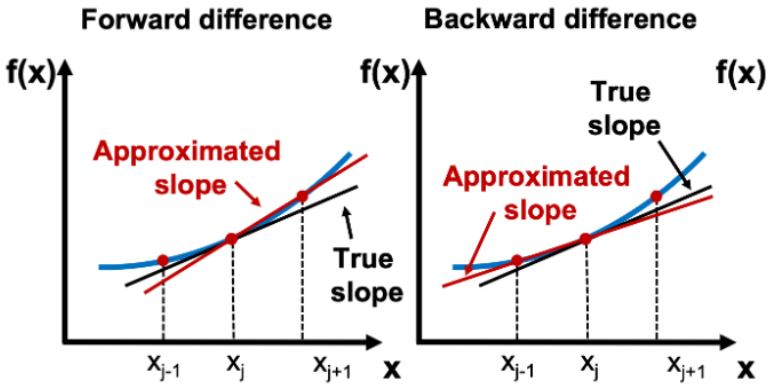
I am unsure which kind of method or function it is used here, because it is only given in the lecture notes [3] without further explanation.

For example, the temperature distribution of a circular dish is calculated by FEM [4]:



## 4. Finite Difference Method

Similarly like Lagrange interpolation, Newton interpolation can be used in Finite Difference Method (FDM). FDM is numerical technique for solving differential equations by approximating derivatives of discrete nodes with finite differences.
The Newton forward and backward interpolation can be used in the finite difference between the nodes




Another possible candidate function for FEM is in the form [3]

$$\frac{k}{\Delta x^2}(a_{i-1}-2a_i+a_{i+1})+f'=m'\ddot{a}_i \quad \text{or} \quad \frac{k}{\Delta x}(a_{i-1}-2a_i+a_{i+1})+F=0 \qquad t>0$$

$$a_0=\underline{a}_0 \quad \text{or} \quad \frac{k}{\Delta x}(a_1-a_0)+F_0=0 \quad \text{and} \quad a_n=\underline{a}_n \quad \text{or} \quad \frac{k}{\Delta x}(a_{n-1}-a_n)+F_n=0 \qquad t>0$$

$$a_i=g_i \text{ and } \dot{a}_i=h_i \qquad t=0$$

# Round 3: Piecewise interpolation

## 1. Comments on the lecture and exercise materials

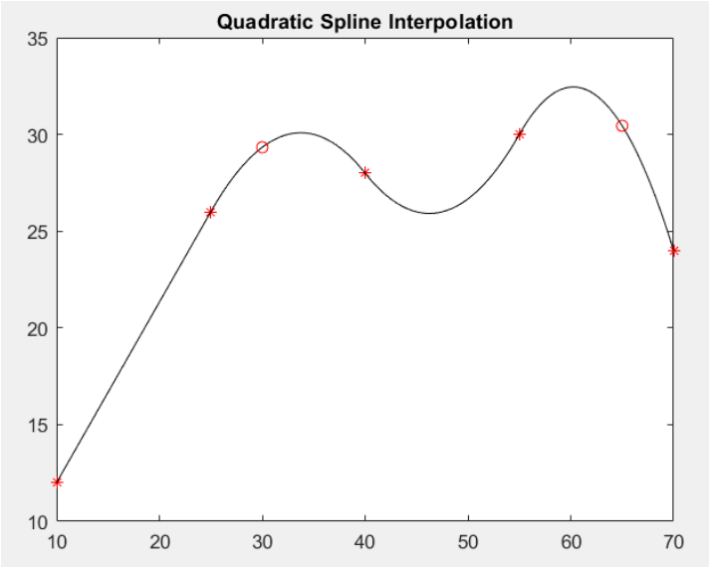
In Exercise 1 of Round 3, the exercise should state the form of g2 and g3, where g2 is piecewise quadratic that has a knot at x = 1 while g3 is a full function of the form ax^3 + bx^2 + cx + d that goes through 2 points 0 and 2 that does not have a knot.
In exercise 3 part (a), the quadhermite function matlab exercise should state that the returning array of 5 elements instead of 6 elements (c10, c11, c12, c20, c21, c22) because the coefficient c20 is actually the c21, because p1(z) = p2(z). In other words, c12 = c20 and one coefficient is redundant

## 2. Reflections on what I have learned

Before then I only heard of the three types of curves, the spline, the B-spline and the Bezier curve. However, the Hermite interpolation seems not to be either of the three lines above. By definition, the Hermite Interpolation is an interpolating polynomial which equals the function and its derivatives up to pth order at (N + 1) data points. Therefore, we would have up to (p + 1)(N + 1) constraints and the interpolating polynomial is of degree (p + 1)(N + 1) – 1, because the number of constraints is equal to the number of unknown coefficients in the interpolating polynomial.

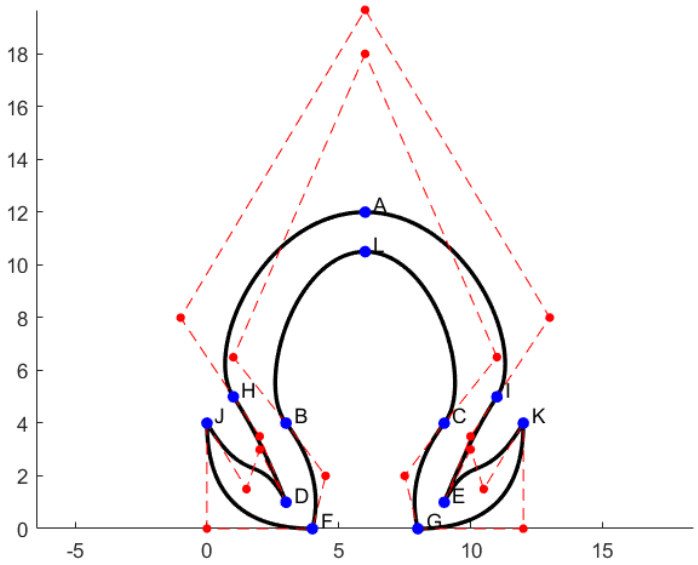
## 3. Spline

When there is a large set of data and a single polynomial is used for interpolation, large errors might occur when a high order polynomial is used. For large number of points a better interpolation can be achieved by using many low-order polynomials [4]. Typically, all of the polynomials are of the same order with different coeffcients in each interval. Interpolation in this way is called piecewise or spline interpolation. There are three types of spline interpolation: linear, quaradtic and cubic splines [5]




## 4. Bezier curves

Bezier curves is a polynomial interpolation where the polynomial includes the two end points. The control points of the Bezier curves will be used to control the curves the of the line and usually the control points do not lie on the Bezier curve. The Bezier curve can have infinitely many control points and the more control points the interpolation has, the more complex the curve becomes. [4]

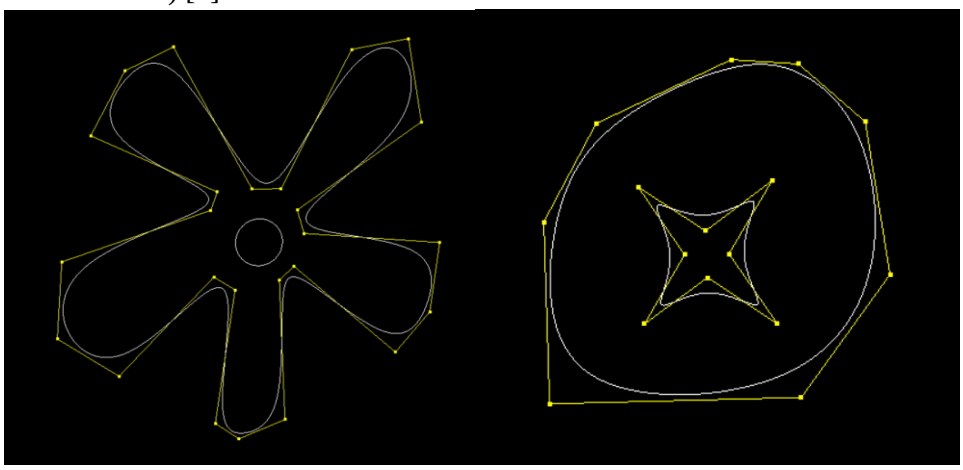



In this drawing of the Greek letter Omega, the blue dots are the end points of the Bezier curve while red dots are the control points of the piecewise Bezier curves [5]. There are second, third and fourth order Bezier curves in this image. For example, from point J to F is the second order, since it has only 1 control point. From point J to D is the third order, where it has 2 control points. Finally, from point

B to C is the fourth order, where it has 3 control points. I found Bezier curve to be a commonly used tool in CAD and vector softwares, such as the Pen tool in Adobe Illustrator software.

## 5. B-Spline

In Bezier curves, all control points will have global control over the interpolating line. However, in B-Spline, we can have different local piecewise interpolation, where the control points only have effects on their respective local part [6]. Therefore, we can use B-Spline to resolve the disadvantage of the Bezier curve if we want to modify locally piecewise polynomials.
For instance, the yellow control points shown in the pictures will result in the piecewise uniform cubic B-spline (the white lines) [6]



## 6. Difference between Spline, B-Spline and Bezier Curves

| Spline | B-Spline | Bezier |
|---|---|---|
| A spline curve goes through a set of coordinate positions (control points) | The B-Spline curves are specified by Bernstein basis function that has limited flexibility. | The Bezier curves can be specified with boundary conditions, with a characterizing matrix or with blending function. |
| It follows the general shape of the curve. | These curves are a result of the use of open uniform basis function. | The curve generally follows the shape of a defining polygon. |
| Application: making designs for automobile bodies, aircraft and spacecraft surfaces and ship hulls. | These curves can be used to construct blending curves. | Usually employed in painting and drawing packages as well as in CAD applications. |
| It possess a high degree of smoothness at the places where the polynomial pieces connect. | The B-Spline allows the order of the basis function and hence the degree of the resulting curve is independent of number of vertices. | The degree of the polynomial defining the curve segment is one less than the number of defining polygon point. |
| A spline curve is a mathematical representation for which it is easy to build an interface that will allow a user to design and control the shape of complex curves and surfaces. | In B-Spline, there is local control over the curve surface and the shape of the curve is affected by every vertex. | It is a parametric curve used in related fields |

Source: Difference between Spline, B-Spline and Bezier Curves at
https://www.geeksforgeeks.org/difference-between-spline-b-spline-and-bezier-curves/

# Round 4: Numerical integration

## 1. Comments on the lecture and exercise materials

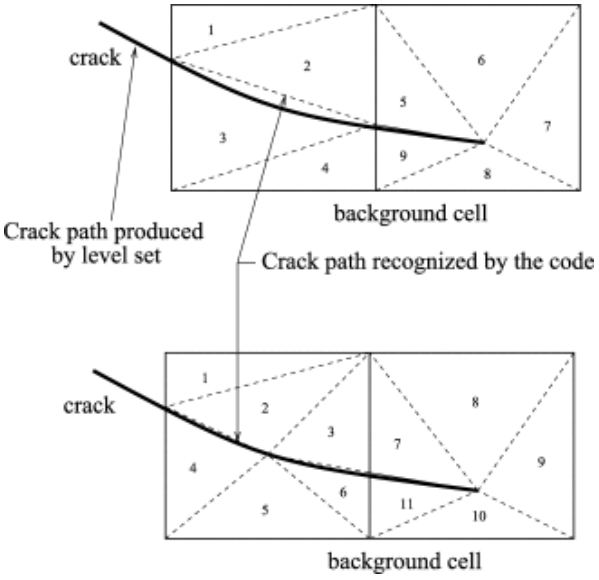
Probably I did not examine the lecture slides carefully, but it appears that well definedness and attributes of the dot product are not mentioned at all in the lecture slides. I think it will be highly recommended if the lecture notes go through these definitions. In Exercise 2b Monte Carlo, the parameter N is not explained at all. I have studied Monte Carlo before so I knew what it refers to, but for students who have just learned about this method, they could be confused. I think the exercises should explain everything in greater details.

## 2. Reflections on what I have learned

I have never known before that a dot product can be interpreted as an integral of two functions. By proving the well-definedness and the attributes of the dot product, the integration can be used in place of the traditional dot product, where vectors are multiplied element wise and the results are summed up. Constructing the second order orthonormal basis is also a meaningful task because it helps me brush up the knowledge about Gram-Schmidt proces that I have learned from the course Linear Algebra. The Guassian quadrature is a new knowledge for me, as it turns out that the integration can be viewed of as a sum of discrete points with varying weights.

## 3. Gaussian Quadrature

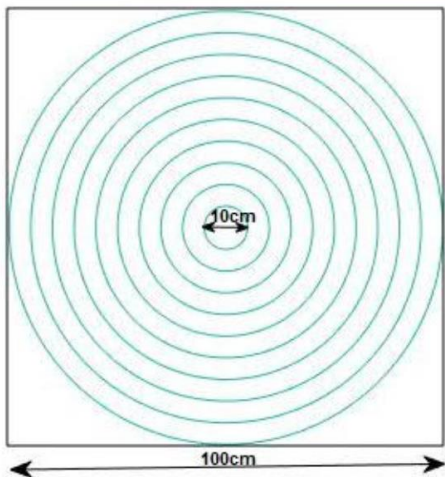
Some knowledge I found personally about Guassian quadrature is that it yields exact values of integrals for polynomials of degree up to 2n – 1 where n is the number of control points. For example, a two point Gaussian rule will produce exact results for polynomials up to degree 3. Gaussian quadrature uses the function values evaluated at a number of interior points and corresponding weights to approximate the integral by a weighted sum [Ryan G. McClarren, in Computational Nuclear Engineering and Radiological Science Using Python, 2018]
In FEM, Gauss quadrature is usually the applied method. However, standard Gauss quadrature cannot be used if the element is crossed by a discontinuity. One integration strategy is based on sub-dividing existing elements into several smaller triangular elements [A. Kayode Coker, in Fortran Programs for Chemical Process Design, Analysis, and Simulation, 1995]



## 4. Monte Carlo

Besides area and volume integration, Monte Carlo can also be used for probability, where the number of points lying inside a region divided by the total number of points generated and the numerical result is normalized.

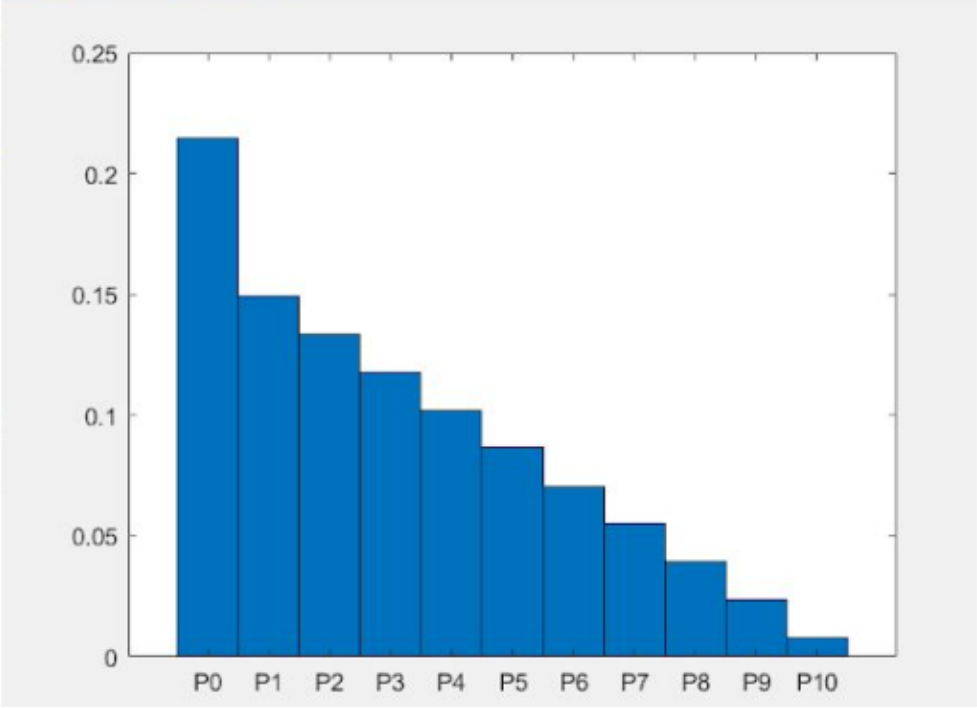
Consider a dart board with 10 concentric circles whose diameters are evenly changing from 10 to 100 cm. Assume that the worst shot is not outside the square inscribing the largest circle. By using the Monte Carlo method, we can find the probability of hitting each of the point from zero (outside the largest circle) to 10 the smallest circle in the middle. The exact solution for each of the points is the ratio of the area corresponded to that point to the area of the square. Credit: this is an exercise from the course Numerical methods in Engineering [5]




Probability of the points using 10000000 randomly generated darts

```
Size of random set: 10000000
Numerical computations for the probability of the points are:
Probability for point 0: 0.21456. True probability is 0.214602
Probability for point 1: 0.14923. True probability is 0.149226
Probability for point 2: 0.13354. True probability is 0.133518
Probability for point 3: 0.11781. True probability is 0.117809
Probability for point 4: 0.1021. True probability is 0.102102
Probability for point 5: 0.086456. True probability is 0.086394
Probability for point 6: 0.070592. True probability is 0.070686
Probability for point 7: 0.055045. True probability is 0.054978
Probability for point 8: 0.039333. True probability is 0.039269
Probability for point 9: 0.02349. True probability is 0.023562
Probability for point 10: 0.0078498. True probability is 0.007854
```

Probability graph for 10000000 darts




Conclusion: with a very large random set of darts generated, the numerical probability will be very close to the true probability.

# Round 5: Initial value problems

## 1. Comments on the lecture and exercise materials

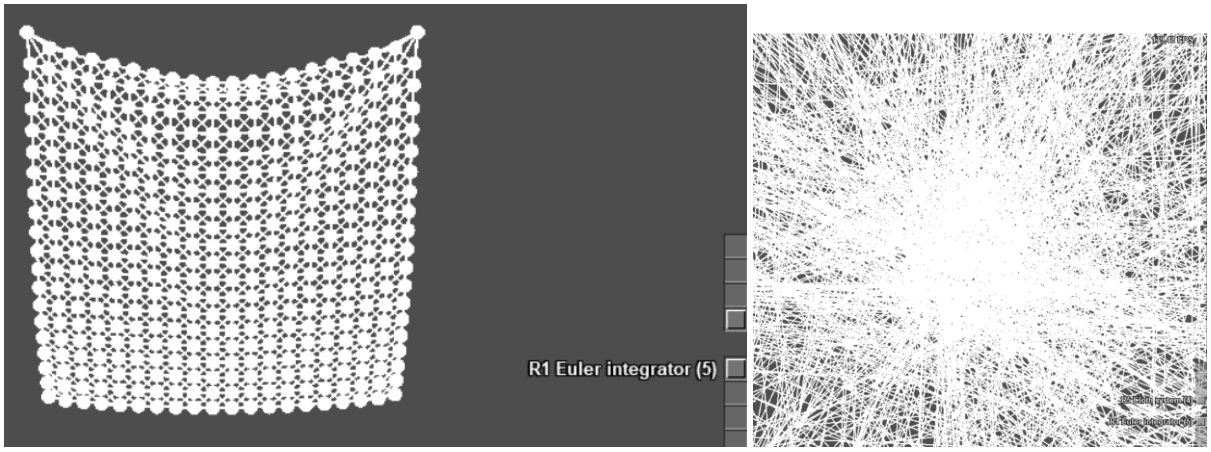
Something that still bothers me about numerical analysis is that, if we can derive the analytical solution of the function and used it to prove other identities, then why would we need to use the Euler's method in order to approximate the function values? I hope the lecture slides should give us a scenario where we must depend on the numerical result because calculating the analytic formula is impossible. Additionally, it seems the lecture slides do not discuss the backward Euler's method

## 2. Reflections on what I have learned

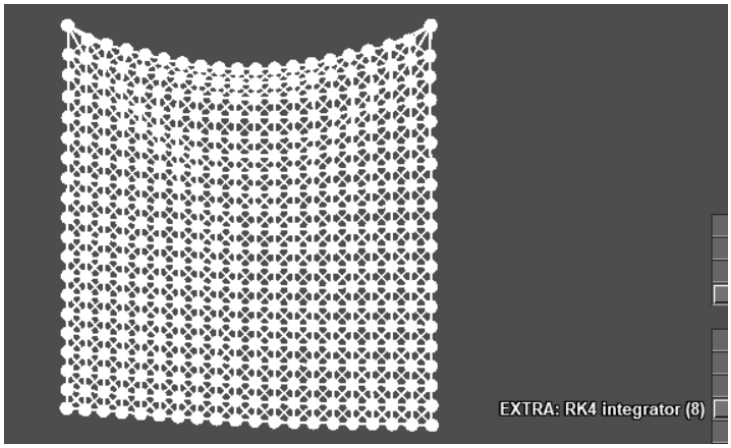
In question 1e, I was surprised to find that the Euler's method approximation can be derived as the function of only initial error and the step size. Changing the parameter error epsilon and the number of step size can also have immense changes in the approximation. For example, with number of step size of only 40, the approximation error explodes to magnitudes of 10^13, while n = 73 up until 75, the error is very stable. I also learned that the Heun's method will almost predict better approximations than the Euler's method thanks to the correction by averaging.

## 3. Euler's method

Euler's method is the most basic explicit method for numerical integration of ordinary differential equations and is the simplest Runge–Kutta method. Euler's method is more preferable than Runge-Kutta method because it provides slightly better numerical results. Its major disadvantage is the possibility of having several iterations that result from a round-error in a successive step [David I. LANLEGE1, Rotimi KEHINDE1, Dolapo A. SOBANKE1, Abdulrahman ABDULGANIYU2, and Umar M. GARBA2]
In computer graphics, the Runge-Kutta methods is often useful to simulate animations and object bound by actual physics. For example, in the below illustrations, the Euler integrator will result in greater errors when the clothes fall down, making it very unstable with large step size [6]

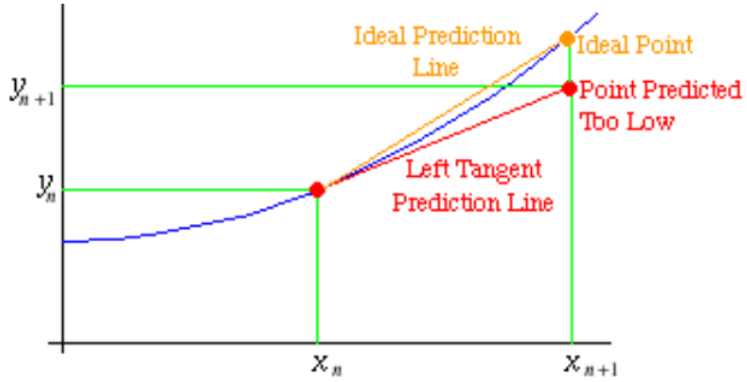



However, for Runge-Kutta method of fourth order, the clothes physics is very stable and can withstand the simulation with large step size

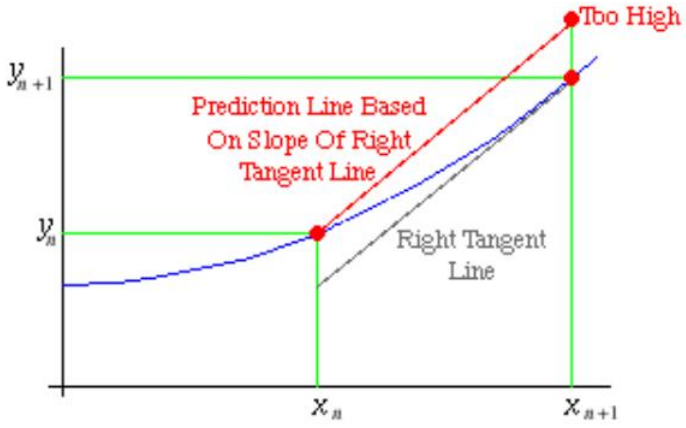



## 4. Heun's method

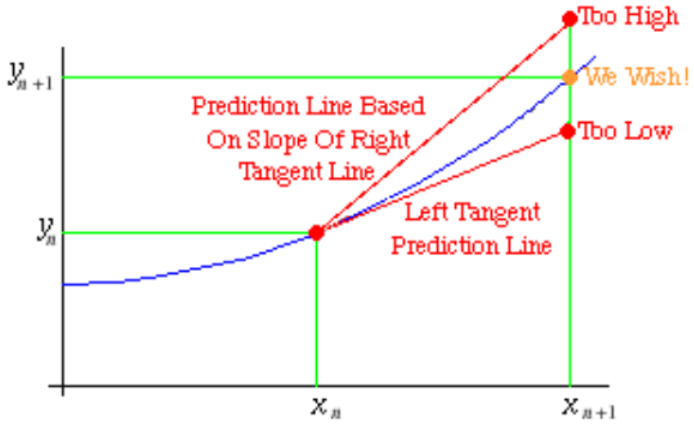
In the real-world applications, numerical solutions do not use the Euler's method but instead, it uses the Heun's method because this method can greatly reduce the error induced by the slope of the tangent line. Therefore, the Heun's Method is sometimes referred to as the Improved Euler Method.
I found a nice page that explains the Heun's Method visually for each step. It is available at
http://calculuslab.deltacollege.edu/ODE/7-C-2/7-C-2-h.html



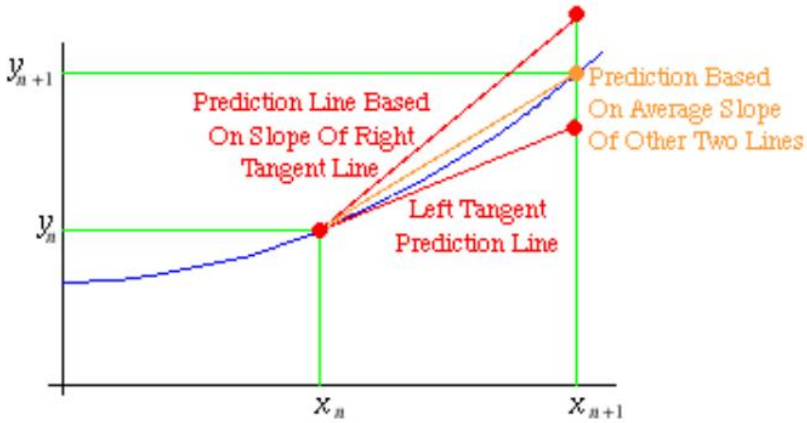

This is the case of the simple Euler's method
Heun's method approach to this problem is to consider the tangent lines to the solution curve at both ends of the interval we are investigating. As we have already established, the tangent line to the curve at the left end-point of the interval is not steep enough for accurate predictions. However if we consider the tangent line to the curve at the right end-point, (assuming we can find it somehow—more on this later,) it has the opposite problem



We can look at the relationship between the error made by both left and right tangent line predictions. One is underestimating the y-coordinate of the next point, and the other is overestimating it.




The point we want appears to lie approximately halfway between these estimates based on tangent lines. We will take the average of the slopes of the left and right tangent lines




## Overall evaluation

This course is a great opportunity for me to brush up my past knowledge from other courses, especially the course Numerical methods in Engineering because my major is the computational engineering. Therefore, I believe if I study more about numerical analysis, it will be remarkably useful for my research in my Bachelor's thesis.

Regarding the exercise session: the teaching assistant is very helpful and I can understand most of the problems after just one session. Honestly, I could not pass this course without attending the exercise session because the exercise instructions are quite vague. Overall, this course has been a great experience for me and I have learned many new useful knowledge.

## References

Many of the knowledge in my learning diary is gathered from other Aalto courses.

[1] Introduction To Optimization MS-C2150
[2] Programming Parallel Computers CS-E4580
[3] Finite Element and Finite Difference Methods COE-C3005
[4] Computer Aided Tools in Engineering ENG-A2001
[5] Numerical Methods in Engineering ENG-A1003
[6] Computer Graphics CS-C31000