Gradient Calculation & Gradient Descent

1. Finite Difference

If we have a function $f(x) \in \mathbb{R}$, $x \in \mathbb{R}$, then the derivative of function in the x_0 is defined as

$$f'(x_0) = \lim_{\epsilon \to 0} \frac{f(x_0 + \epsilon) - f(x_0)}{\epsilon}$$

or equivalently (symetric derivative)

$$f'(x_0) = \lim_{\epsilon \to 0} \frac{f(x_0 + \epsilon) - f(x_0 - \epsilon)}{2\epsilon}$$

Using this standard derivation we can numerically compute the function derivation.

2. Gradient computation via Projection

First of all lets consider the Span of the set $P = \{p_i(x) |\ p_i(x) = x^i,\ i \in \mathbb{N}_0,\ x \in \mathbb{R}\}$. Span is the linear combination of the set elements, i.e. $\mathcal{P} = \text{Span}[P] = \{\sum_0^N \alpha_i p_i(x) |\ \forall \alpha_i \in \mathbb{R},\ N \in \mathbb{N}\}$. We can prove that $\mathcal{P}^N = (\mathcal{P}, +, \cdot)$ with

$$\begin{aligned} + :&\quad \mathcal{P} \times \mathcal{P} \to \mathcal{P} \\ &\quad \sum \alpha_i p_i(x) + \sum \beta_i p_i(x) = \sum_0^N (\alpha_i + \beta_i) p_i(x) \\ \cdot :&\quad \mathbb{R} \times \mathcal{P} \to \mathcal{P} \\ &\quad \lambda \cdot \sum_0^N \alpha_i p_i(x) = \sum_0^N \lambda \alpha_i p_i(x) \end{aligned}$$

is the vector space
1) Closure $\forall P_1, P_2 \in \mathcal{P} :\ P_1 + P_2 \in \mathcal{P}$
$\square$ $\alpha_i \in \mathbb{R},\ \beta_i \in \mathbb{R} \Rightarrow (\alpha_i + \beta_i) \in \mathbb{R} \Rightarrow \sum (\alpha_i + \beta_i) p_i(x) \in \mathcal{P}$ $\blacksquare$

2) Commutativity $\forall P_1, P_2 \in \mathcal{P} :\ P_1 + P_2 = P_2 + P_1$
$\square$ $\alpha_i, \beta_i \in \mathbb{R} \Rightarrow \alpha_i + \beta_i = \beta_i + \alpha_i \Rightarrow P_1 + P_2 = P_2 + P_1$ $\blacksquare$
3) Associativity $\forall P_1, P_2, P_3 \in \mathcal{P} :\ P_1 + (P_2 + P_3) = (P_1 + P_2) + P_3$
$\square$ Similarly $\blacksquare$

4) Neutral element $\exists P_0 \in \mathcal{P}\, \forall P_1 \in \mathcal{P} :\ P_1 + P_0 = P_0 + P_1 = P_1$
$\square$ $\sum \alpha_i p_i(x) + \sum \beta_i p_i(x) = \sum (\alpha_i + \beta_i) p_i(x) \Rightarrow \beta_i = 0 \forall i \in \mathbb{N}_0$, i.e. $P_0 = 0 + 0x + 0x^2 + ... = 0$ $\blacksquare$

5) Inverse element $\forall P_1 \in \mathcal{P}\, \exists P_{-1} \in \mathcal{P} :\ P_1 + P_{-1} = P_{-1} + P_1 = P_0$
$\square$ $\alpha_i + \alpha_{-i} = 0\ \forall i \in \mathbb{N}_0 \Rightarrow \alpha_{-i} = -\alpha_i$ $\blacksquare$

6) Distributivity respect to $\cdot$

- $\forall \lambda \in \mathbb{R}, \forall P_1, P_2 \in \mathcal{P} :\ \lambda \cdot (P_1 + P_2) = \lambda \cdot P_1 + \lambda \cdot P_2$
  $\square$ $\lambda \cdot (\alpha_i + \beta_i) = \lambda \cdot \alpha_i + \lambda \cdot \beta_i$ $\blacksquare$
- $\forall \lambda, \mu \in \mathbb{R},\ P_1 \in \mathcal{P} :\ (\lambda + \mu) \cdot P_1 = \lambda \cdot P_1 + \mu \cdot P_1$
  $\square$ Similarly $\blacksquare$

7) Associativity respect to $\cdot$, $\forall \lambda, \mu \in \mathbb{R},\ P_1 \in \mathcal{P} :\ \lambda \cdot (\mu \cdot P_1) = (\lambda \mu) \cdot P_1$
$\square$ $\blacksquare$

so $\mathcal{P}^N$ is the vector space. Obviously that the set $\{1, x, x^2, ..., x^N\}$ is the non orthogonal basis of $\mathcal{P}^N$ relative to the following inner product *

$$\begin{aligned} \langle \cdot, \cdot \rangle :&\quad \mathcal{P} \times \mathcal{P} \to \mathbb{R} \\ &\quad \langle p_i(x), p_j(x) \rangle := \int_0^1 p_i(x) p_j(x)\, dx \end{aligned}$$

□ Linearity follows from the linearity of the integration operation. Positive-definiteness follows from the fact, that definite integral of positive function has a positive value ■.
$\langle\cdot,\cdot\rangle$ induced *p norm* $||\cdot||_2$ is

$$\begin{aligned} ||\cdot||_2: \quad & \mathcal{P} \to \mathbb{R} \\ & p(x) \mapsto ||p(x)||_2 = \left(\int_0^1 |p(x)|^2\,dx\right)^{1/2} \end{aligned}$$

Using the Gram-Schmidt procedure we can find the orthogonal basis, for example for $N = 2$ we will have $\{\hat{p}_0(x) = 1, \hat{p}_1(x) = 2\sqrt{3}\left(x - \frac{1}{2}\right), \hat{p}_2(x) = 6\sqrt{5}\left(\frac{1}{6} - x + x^2\right)\}$ .But we will we can proceed from the condition of orthogonality of the projection $\langle f(x) - \pi_{\mathcal{P}^N}(f(x)), P\rangle = 0,\ \forall P \in \mathcal{P}$ of $f(x)$, and find the orthogonal projection $\pi_{\mathcal{P}^N}(f(x))$. We can prove this condition if we have vector subspace $\mathcal{P}^N$ spanned by $\{\hat{p}_0(x), \hat{p}_1(x), ..., \hat{p}_N(x)\}$ orthonormal vector set, then

$$\begin{aligned} \langle f(x) - \pi_{\mathcal{P}^N}(f(x)), P\rangle = & \qquad (\forall P \in \mathcal{P},\ \text{i.e.} \pi_{\mathcal{P}^N}(f(x)) = \alpha_i\hat{p}_i(x),\ P = \beta_j\hat{p}_j(x)) \\ & = \beta_j\langle f(x), \hat{p}_j(x)\rangle - \beta_j\langle \alpha_i\hat{p}_i(x), \hat{p}_j(x)\rangle \\ & = \beta_j\alpha_j - \beta_j\alpha_i\langle \hat{p}_i(x), \hat{p}_j(x)\rangle \\ & = \beta_j\alpha_j - \beta_j\alpha_i\delta_{ij} \\ & = \beta_j\alpha_j - \beta_j\alpha_j = 0 \end{aligned}$$

Using this condition, we get

$$\begin{aligned} \int_0^1 (f(x) - \pi_{\mathcal{P}^N}(f(x)))\,x^i\,dx = 0 & \\ \int_0^1 \pi_{\mathcal{P}^N}(f(x))\,x^i\,dx = \int_0^1 f(x)\,x^i\,dx & \qquad \pi_{\mathcal{P}^N}(f(x)) = \sum_{k=0}^{N}\alpha_k x^k \\ \sum_{k=0}^{N}\alpha_k\int_0^1 x^k x^i\,dx = \int_0^1 f(x)\,x^i\,dx & \\ \sum_{k=0}^{N}\frac{\alpha_k}{i+k+1} = \int_0^1 f(x)\,x^i\,dx & \end{aligned}$$

$\alpha_k,\ k \in \mathbb{N}_0$ is the coordinates of projection which we want to find. Considering that $i = 0, 1, \ddots, N$ we can write equation in the matrix form

$$\begin{aligned} A &:= \sum_{i=0}^{N}\sum_{k=0}^{N}\frac{1}{i+k+1} = \left\{\frac{1}{i+k+1}\,\middle|\, i,k \in \mathbb{N}_0\right\} \qquad (N+1)\times(N+1) \text{ Matrix} \\ \alpha &:= \alpha_k,\ k \in \mathbb{N}_0 \qquad n \text{ vector} \\ b &:= \int_0^1 f(x)x^i\,dx,\ i \in \mathbb{N}_0 \qquad n \text{ vector} \end{aligned}$$

So we have equation $A\alpha = b \Rightarrow \alpha = A^{-1}b$ and finally we get the following expression for the projection:

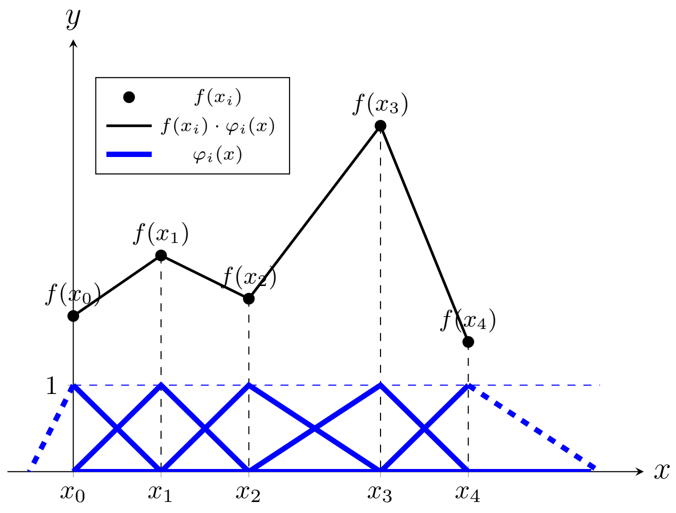
$$f(x) \approx \pi_{\mathcal{P}^N}(f(x)) = \sum_{i=0}^{N}(A^{-1}b)_i\,x^i$$

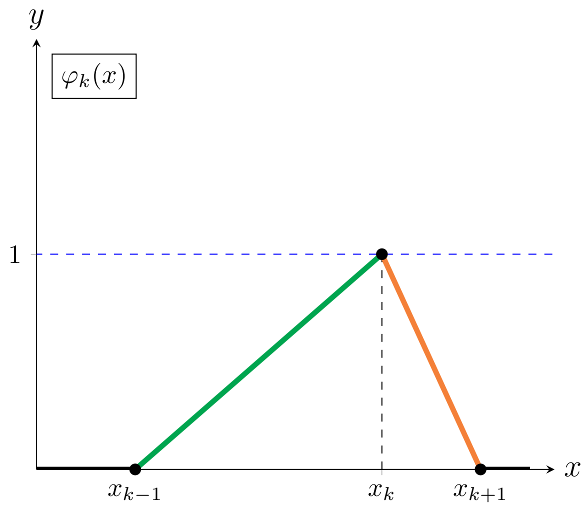
$$A = \begin{pmatrix} 1 & \frac{1}{2} & \frac{1}{3} & \cdots & \frac{1}{N+1} \\ \frac{1}{2} & \frac{1}{3} & \frac{1}{4} & \cdots & \frac{1}{N+2} \\ \vdots & & & & \vdots \\ \frac{1}{N+1} & \frac{1}{N+1} & \frac{1}{N+2} & \cdots & \frac{1}{2N+1} \end{pmatrix}, \quad b = \begin{bmatrix} \int_0^1 f(x)\,dx \\ \int_0^1 xf(x)\,dx \\ \int_0^1 x^2 f(x)\,dx \\ \vdots \\ \int_0^1 x^N f(x)\,dx \end{bmatrix}$$

So we can find the derivative of $f(x)$

$$\boxed{\frac{d}{dx}f(x) \approx \sum_{i=0}^{N} i\,(A^{-1}b)_i\,x^{i-1}}$$

We need compute the vector b, for this we can use the approximation of the $f(x)$ by the 'cap' functions $\varphi_k(x)$





$$\varphi_k(x) = \begin{cases} \dfrac{x - x_{k-1}}{x_k - x_{k-1}} & \text{in } [x_{k-1}, x_k] \\ \dfrac{x_{k+1} - x}{x_{k+1} - x_k} & \text{in } [x_k, x_{k+1}] \\ 0 & \text{Else} \end{cases}$$

We can represent $f(x)$ in the form $f(x) \approx \sum_{k=0}^{n} f(x_k)\varphi_k(x)$ therefore

$$\int_0^1 f(x)x^i\,dx = f(x_0)\int_{x_0}^{x_1}\varphi_0(x)x^i\,dx + \sum_{k=1}^{n-1}\left(f(x_k)\int_{x_{k-1}}^{x_{k+1}}\varphi_k(x)x^i\,dx\right) + f(x_n)\int_{x_{n-1}}^{x_n}\varphi_n(x)x^i\,dx$$

after integration we get

$$\begin{aligned}\int_0^1 f(x)x^i\,dx =& \frac{f(x_0)}{(i+1)(i+2)}\frac{x_1^{i+2} - x_0^{i+1}[x_1(i+2) - x_0(i+1)]}{x_1 - x_0} + \frac{f(x_n)}{(i+1)(i+2)}\frac{x_{n-1}^{i+2} - x_n^{i+1}\,[x_{n-1}(i+2) - x_n(i+1)]}{x_n - x_{n-1}} \\ &+ \sum_{k=1}^{n-1}\frac{f(x_k)}{(i+1)(i+2)}\left(\frac{x_{k-1}^{i+2} - x_k^{i+1}\,[x_{k-1}(i+2) - x_k(i+1)]}{x_k - x_{k-1}} + \frac{x_{k+1}^{i+2} - x_k^{i+1}\,[x_{k+1}(i+2) - x_k(i+1)]}{x_{k+1} - x_k}\right) = \\ &\boxed{\begin{aligned} =& \sum_{k=1}^{n-1}\frac{f(x_k)}{(i+1)(i+2)}\frac{x_{k-1}^{i+2} - x_k^{i+1}\,[x_{k-1}(i+2) - x_k(i+1)]}{x_k - x_{k-1}} + \\ &+ \sum_{k=0}^{n}\frac{f(x_k)}{(i+1)(i+2)}\frac{x_{k+1}^{i+2} - x_k^{i+1}\,[x_{k+1}(i+2) - x_k(i+1)]}{x_{k+1} - x_k}\end{aligned}}\end{aligned}$$

For some reason, we may need to calculate an integral with a non-zero x_0 lower bound. Vector b will not change in this case but we must modify the matrix A

$$\tilde{A} = A - x_0^A \circ A, \qquad \circ \text{ is the Hadamard multiplication}$$

To calculate the vector b, we can use other numerical methods for finding integrals, for example using the so called trapezoidal rule.

Python:

The function for compute derivative by definition:

```python
def der_finit(f,x,n):
  """One variable scalar function derivative by deffinition"""


  if isinstance(x, np.ndarray) == True:
    pass
  else:
    x=np.linspace(x,1,100)

  return (f(x)[1:] - f(x)[:-1])/(x[1:] - x[:-1])
```

function to calculate the vector b (integrals are calculated using $\phi_k(x)$ 'cap' functions)

```python
def b_cap(f,x,n):
  """Vector b for determining polynomial projection 'coordinates'.
  to find the integrals, the 'cap' function is used """

  i = np.arange(0,n+1)
  j = i.reshape(-1,1)
  x = np.tile(x.reshape(-1,1),(1,len(i)))

  Ik1 = (((f[1:]))/((j+1)*(j+2)))*((np.power(x[:-1],(i+2)) - np.power(x[1:],(i+1))*(x[0:-1]*(i+2) - x[1:]*(i+1)))/(x[1:] - x[:-1])).T
  Ik2 = (((f[:-1]))/((j+1)*(j+2)))*((np.power(x[1:],(i+2)) - np.power(x[:-1],(i+1))*(x[1:]*(i+2) - x[:-1]*(i+1)))/(x[1:] - x[:-1])).T

  return np.sum(np.concatenate((Ik1, Ik2), axis=1),axis=1)
```

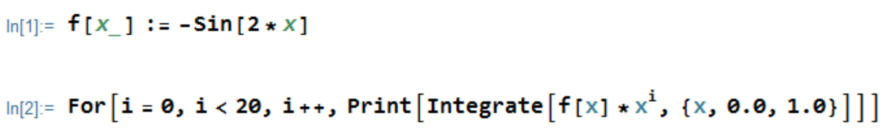
function to calculate the vector b (integrals are calculated using trapezoidal rule)

```python
def b_trap(f,x,n):

  i = np.arange(0,n+1)
  x = np.tile(x.reshape(-1,1),(1,len(i)))

  return np.sum(((x[1:]-x[:-1]).T*((np.power(x,i).T*F).T[:-1] +(np.power(x,i).T*F).T[1:]).T/2) , axis = 1)
```

We can compare with the results that the sipson function of the scipy package gives

```python
def b_sci(f,x,n):

  b = np.asarray([integrate.simpson(f*x**i, x) for i in range(n+1)])

  return b
```

Or/And we can compare with the result which gives Mathematica



```
In[1]:= f[x_] := -Sin[2*x]

In[2]:= For[i = 0, i < 20, i++, Print[Integrate[f[x]*x^i, {x, 0.0, 1.0}]]]
```

Matrix $\tilde{A}$ computing function

```python
def Ainv_matrix(x0,n):

  A = np.full((n+1, n+1), np.arange(1,n+2,1)) + np.arange(0,n+1,1).reshape(-1,1)
  P = np.np.powerer(x0,A)
  M = np.multiply(1/A,P) # Hadamard multiplication
  A = 1/A - M
  invA = np.linalg.inv(A)

  return invA
```

The comparison shows that b_cap correctly calculates the vector b for different values of N.

And finaly the function which compute derivative by projection

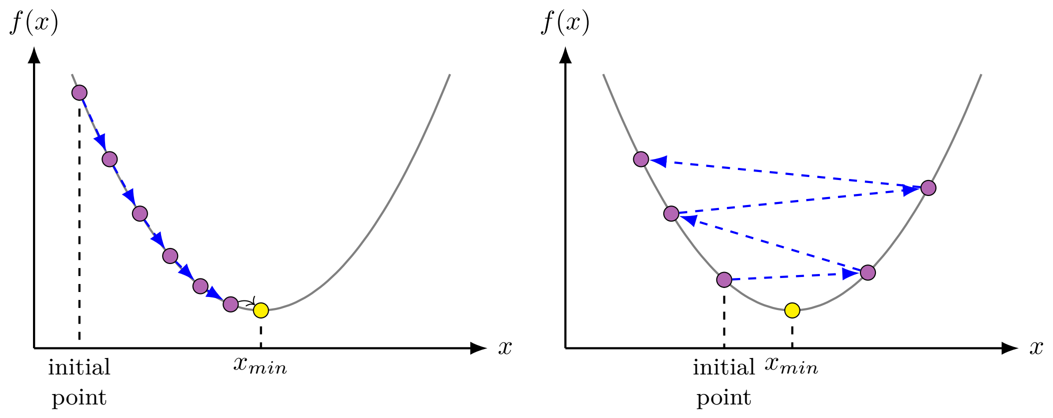
```python
def der_proj(f,x,n):

  if isinstance(x, np.ndarray) == True:
    pass
  else:
    x=np.linspace(x,1,100)

  i = np.arange(1,n+1)
  X = x.reshape(-1,1)
  X = np.tile(X,(1,len(i)))

  return np.sum(i*((Ainv_matrix(x[0],n)@b_sci(f(x),x,n))[1:])*np.power(X,i-1), axis = 1)
```

3. Gradient descent

Gradient descent is a iterative process for finding the minimum of some function. The process starts from the some arbitrarily chosen point x_0 and move in the opposite direction from the gradient. At the same time, we can use the coefficient that specifies the step length α called *learning rate.*

$$x_1 = x_0 - \alpha \nabla_x f(x)\big|_{x_0}$$

we iteratively continue the process until a certain condition is hold, for example $|x_{n+1} - x_n| < \epsilon$, where epsilon is the some threshold. We choose the parameters α and ϵ by hand and based on the selected values, different scenarios may arise. For example if learning rate too small, then the process will converge very slowly and vice versa, if it is too large, the process may not converge. The following plots schematically illustrate these scenarios:

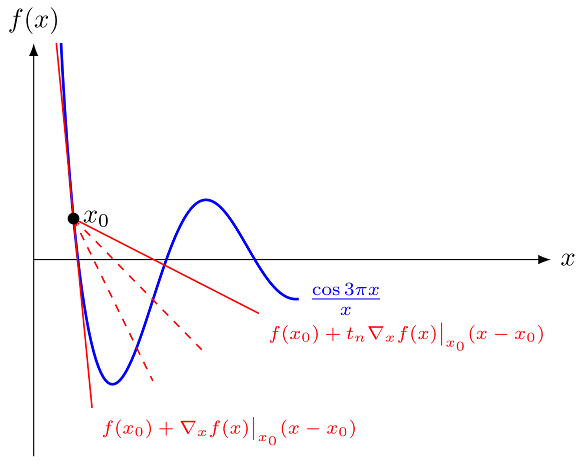


There are various modifications of the simple gradient descent algorithm. For example, we can dynamically change the learning rate α along iterations based on some conditions. But we will do it differently, we modify the simple algorithm as follows, we 'break' the tangent line, which is determined by the derivative at a particular point and tangent equation has the following form

$$f(x) = f(x_0) + \nabla_x f(x)\big|_{x_0}(x - x_0)$$

we'll take many paths based on which we will change the x

$$f(x) = f(x_0) + \alpha t_i \nabla_x f(x)\big|_{x_0}(x - x_0), \qquad i \in \mathbb{N},\, t_i \in \mathbb{R}$$


$f(x)$
x
x_0
$\frac{\cos 3\pi x}{x}$
$f(x_0) + t_n \nabla_x f(x)\big|_{x_0}(x - x_0)$
$f(x_0) + \nabla_x f(x)\big|_{x_0}(x - x_0)$

therefore x will change as follows

$$x_{n+1} = x_n - \alpha t_i \nabla_x f(x)\big|_{x_n}$$

and from the set of points $\{x_{n+1}\}$, we select that point as a new starting point, on which the function $f(x)$ has a smaller value compared to the rest.

Python:

```python
def grad_descent(f,div,start,w,t,e,n=0):

  delta=1
  steps=10

  while delta > e and steps > 0:

    x = start - w*t*(div(f,start,n)[0])

  if np.all(x) < 0 or np.all(x) > 1:

    print('---Reset start point---')
    x = np.random.uniform(0,1)
    steps -= 1

  delta = abs(x[np.argmin(f(x))]-start)
  start = x[np.argmin(f(x))]
  print(delta)
  print(start)

  return start
```