



Sparse and Robust Signal Reconstruction

Sandra V. B. Jardim*

Tomar Polytechnic Institute, Department of Information Technology, Quinta do Contador, Estrada da Serra - Tomar, Portugal.

Abstract

Many problems in signal processing and statistical inference are based on finding a sparse solution to an undetermined linear system. The reference approach to this problem of finding sparse signal representations, on overcomplete dictionaries, leads to convex unconstrained optimization problems, with a quadratic term ℓ_2 , for the adjustment to the observed signal, and a coefficient vector ℓ_1 -norm. This work focus the development and experimental analysis of an algorithm for the solution of ℓ_q - ℓ_p optimization problems, where $p \in]0, 1] \wedge q \in [1, 2]$, of which ℓ_2 - ℓ_1 is an instance. The developed algorithm belongs to the majorization-minimization class, where the solution of the problem is given by the minimization of a progression of majorizers of the original function. Each iteration corresponds to the solution of an ℓ_2 - ℓ_1 problem, solved by the projected gradient algorithm. When tested on synthetic data and image reconstruction problems, the results shows a good performance of the implemented algorithm, both in compressed sensing and signal restoration scenarios.

Keywords: Sparse signal representation, Convex relaxation, ℓ_2 - ℓ_1 optimization, Compressed sensing, Majorization-minimization algorithms, Quadratic programming, Gradient projection algorithms.
2010 MSC: 93C42, 94A12.

1. Introduction

In general, sparse approximation problems have been of great interest given its wide applicability both in signal and image processing field as in statistical inference contexts, where many of the problems to be solved involve the undetermined linear systems sparse solutions determination.

The literature on sparsity optimization is rapidly increasing (see (Zarzer, 2009; Donoho, 2006; Candes & Tao, 2005; Wright *et al.*, 2009) and references therein). More recently sparsity techniques are also receiving increased attention in the optimal control community (Stadler, 2009; Casas *et al.*, 2012; Herzog *et al.*, 2012).

Given an input signal \mathbf{y} , sparse approximation problems resolution aims an approximated signal determination \mathbf{x} through a linear combination of elementary signals, which are, for several

*Corresponding author

Email address: sandra.jardim@ipt.pt (Sandra V. B. Jardim)

current applications, extracted from a set of signals not necessarily linearly independent. A preference for sparse linear combinations is imposed by penalizing nonzero coefficients. The most common penalty is the number of elementary signals that participate in the approximation. According to the context in which they operate and the objective to be achieved, sparse approximation problems can be formulated in different ways. In this sense, the problem domain must justify the linear model, the elementary signals choice and the sparsity criterion to be adopted.

On account of the combinatorial nature of the sparse approximation problems, which is due to the presence of the *quasi*-norm ℓ_0 of the coefficients vector to be estimated, these problems have a difficult computational resolution. In general, these optimization problems are NP-hard problems (Davis *et al.*, 1997; Natarajan, 1995). One of the most common approach to overcome this difficulty is the convex relaxation, introduced by Claerbout et al. (Claerbout & Muir, 1973), of the original problem, where the *quasi*-norm ℓ_0 is replaced by the norm ℓ_1 , which is a convex function. A classic example of this kind of problems is the determination of sparse representations on overcomplete dictionaries, where the reference approach leads to unconstrained convex optimization problems, which involves a quadratic term ℓ_2 of adjustment to the observed signal \mathbf{y} , and a ℓ_1 norm of the coefficient vector to be estimated \mathbf{x} . In this sense, it is notorious the interest shown by the scientific community in the development of methods leading to the resolution of the unconstrained convex optimization problem

$$\min_{\mathbf{x}} \frac{1}{2} \|\mathbf{y} - \phi \mathbf{x}\|_2^2 + \lambda \|\mathbf{x}\|_1, \quad (1.1)$$

where ϕ represents the overcomplete dictionary synthesis matrix; if $\mathbf{x} \in \mathbb{R}^n$ and $\mathbf{y} \in \mathbb{R}^k$, ϕ is a $k \times n$ matrix. The nonnegative parameter λ states a compromise between the approximation mean squared error and his sparsity level.

The above optimization problem, and related ones, arises in several applications, such as the *Basis Pursuit* and *Basis Pursuit Denoising* criterions (Chen *et al.*, 2001) and *Compressed Sensing* (Donoho, 2006).

The optimization problem represented in (1.1) is an instance of the general class of the optimization problems $\ell_q - \ell_p$, where q and p can assume values in the range $]0, 2]$. It is important to stress that the data term generalization to a ℓ_q norm, instead of the ℓ_2 norm, gives to the approximation criterion statistical strength features (when $q < 2$) (Huber & Ronchetti, 2009), making it less permeable to spurious observations (*outliers*). On the other hand, when it comes to generalization of the coefficient term to be estimated to a ℓ_p norm, instead a ℓ_1 norm, and considering $p < 1$, we walk toward the original combinatorial problem resolution.

In this paper is presented an algorithm that aims the resolution of the general class optimization problems $\ell_q - \ell_p$, where $p \in]0, 1] \wedge q \in [1, 2]$. The developed algorithm belong to the majorization-minimization class (Hunter & Lange, 2004), where the problem is solved in an iterative way, through the minimization of a majorizers sequence of the original function. Each upper bound function corresponds to a $\ell_2 - \ell_1$ problem, where each one of these problems is formulated as a quadratic programming problem and solved through the gradient projection algorithm (Figueiredo *et al.*, 2007b).

2. Generalized Optimization Problem

The unconstrained optimization problem represented in (1.1) is the convex relaxation of the subset selection problem, checking to be a major problem in many application areas. Due to its high applicability, there has been considerable effort made by the scientific community, regarding techniques and algorithms development for its resolution. Among the different proposed algorithms, are homotopy algorithms (Turlach, 2005; Efron *et al.*, 2004; Malioutov *et al.*, 2005), the ones based on interior-point methods (Turlach *et al.*, 2005; Chen *et al.*, 2001), the majorization-minimization class algorithms (Figueiredo & Nowak, 2003, 2005; Figueiredo *et al.*, 2007a) and the gradient projection algorithm (Figueiredo *et al.*, 2007b).

In this problem, the objective function composed by two terms, one of which being a quadratic term ℓ_2 of adjustment to the observed signal \mathbf{y} , and the other the ℓ_1 norm of the coefficient vector to estimate \mathbf{x} . Recall that the ℓ_1 norm arises by replacing the *quasi*-norm ℓ_0 , at the convex relaxation of the original convex optimization problem.

As stated above, the optimization problem $\ell_2 - \ell_1$ is an instance of the optimization problems $\ell_q - \ell_p$ general class, where $p \in]0, 1] \wedge q \in [1, 2]$.

Since the purpose of this work is to achieve the solution of the general class optimization problems $\ell_q - \ell_p$, for $p \in]0, 1] \wedge q \in [1, 2]$, let's consider the generalization of the unconstrained convex optimization problem (1.1), and define the function $L(\mathbf{x})$

$$L(\mathbf{x}) = \frac{1}{2} \|\mathbf{y} - \phi \mathbf{x}\|_q^q + \lambda \|\mathbf{x}\|_p^p, \quad (2.1)$$

where $\mathbf{x} \in \mathbb{R}^n$, $\mathbf{y} \in \mathbb{R}^k$, ϕ is the synthesis matrix of a dictionary D (of dimension $k \times n$), $\lambda \geq 0$, $p \in]0, 1]$ and $q \in [1, 2]$.

3. Majorization-Minimization Method for the Resolution of the Generalized Optimization Problem

3.1. Objective Function

There are several algorithms for the resolution of the optimization problem (1.1), such as Matching Pursuit (MP), Orthogonal Matching Pursuit (OMP), Homotopy, Least Absolute Shrinkage and Selection Operator (LASSO) and Gradient Projection (GPSR). In this work is developed a majorization-minimization (MM) method, for the resolution of the optimization problem $\min_{\mathbf{x}} L(\mathbf{x})$, where the optimization problem (1.1) is solved using the Barzilai-Borwein Gradient Projection algorithm for sparse reconstruction (GPSR-BB) (Figueiredo *et al.*, 2007b). This choice results from an analysis of the results obtained for different algorithms, which can be found in (Jardim, 2008). Since GPSR-BB algorithm aims to solve the optimization problem (1.1), it is necessary to establish a relation between this and the optimization problem that results from the minimization of the function $L(\mathbf{x})$ (2.1).

We can observe that the function $L(\mathbf{x})$ is the sum of the functions $L_1(\mathbf{x})$ and $L_2(\mathbf{x})$, where

$$L_1(\mathbf{x}) = \frac{1}{2} \|\mathbf{y} - \phi \mathbf{x}\|_q^q \quad \text{and} \quad (3.1)$$

$$L_2(\mathbf{x}) = \lambda \|\mathbf{x}\|_p^p. \quad (3.2)$$

Knowing that $\|w\|_r^r = \sum_i |w_i|^r$, we can define the function

$$f(z, p) = |z|^p. \quad (3.3)$$

Figures (1(a)) and (1(b)) are graphical representations of the function $f(z, p)$ for different values of p .

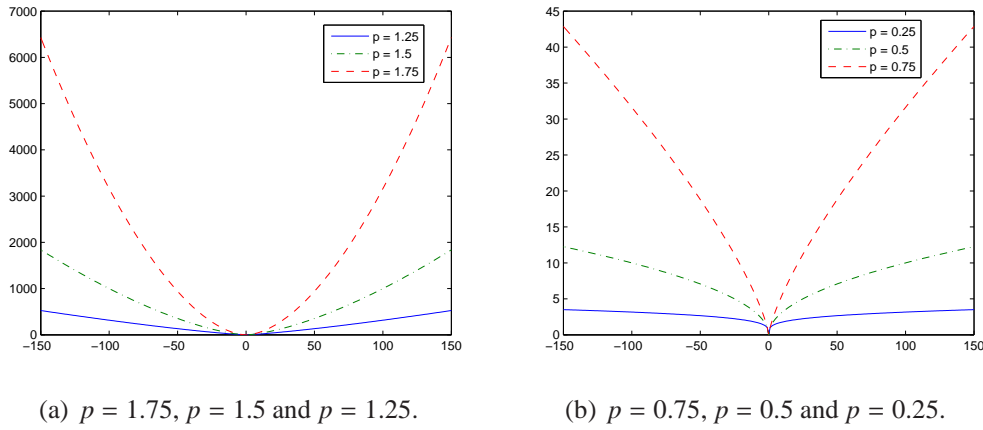


Figure 1. Function $f(z, p) = |z|^p$ for values of p greater and smaller than 1.

With this function it is possible to write (3.1) and (3.2) as

$$L_1(\mathbf{x}) = \frac{1}{2} \sum_{i=1}^k f(y_i - (\phi\mathbf{x})_i, q). \quad (3.4)$$

and

$$L_2(\mathbf{x}) = \lambda \sum_{i=1}^n f(x_i, p). \quad (3.5)$$

3.1.1. The Majorizer Function

By the analysis of the figures (1(a)) and (1(b)) we can verify that it is possible to determine for the function $f(\mathbf{z}, p)$ (so to the $L_1(\mathbf{x})$ and $L_2(\mathbf{x})$ functions), and for some value \mathbf{z}' , majorizers functions. This opens the door to the use of majorization-minimization algorithms to the resolution of the optimization problem $\min_{\mathbf{x}} L(\mathbf{x})$, where $L(\mathbf{x})$ is the function given by (2.1). So, it is necessary to define a Q function such that

$$L(\mathbf{x}) \leq Q(\mathbf{x}|\hat{\mathbf{x}}^{(t)}), \quad \forall_{\mathbf{x} \neq \hat{\mathbf{x}}^{(t)}} \quad (3.6)$$

$$L(\hat{\mathbf{x}}^{(t)}) = Q(\hat{\mathbf{x}}^{(t)}|\hat{\mathbf{x}}^{(t)}), \quad (3.7)$$

i.e., $Q(\mathbf{x}|\hat{\mathbf{x}}^{(t)})$ is a function of \mathbf{x} that majorizes (i.e., upper bounds) $L(\mathbf{x})$.

Recalling that $L(\mathbf{x}) = L_1(\mathbf{x}) + L_2(\mathbf{x})$, where $L_1(\mathbf{x})$ and $L_2(\mathbf{x})$ are given by (3.4) and (3.5) respectively, let's consider the majorizer functions

$$Q_1(\mathbf{x}|\hat{\mathbf{x}}^{(t)}) \geq L_1(\mathbf{x}) \quad \text{and} \quad Q_1(\hat{\mathbf{x}}^{(t)}|\hat{\mathbf{x}}^{(t)}) = L_1(\hat{\mathbf{x}}^{(t)}) \quad (3.8)$$

and

$$Q_2(\mathbf{x}|\hat{\mathbf{x}}^{(t)}) \geq L_2(\mathbf{x}) \quad \text{and} \quad Q_2(\hat{\mathbf{x}}^{(t)}|\hat{\mathbf{x}}^{(t)}) = L_2(\hat{\mathbf{x}}^{(t)}). \quad (3.9)$$

Given the MM algorithm properties (Hunter & Lange, 2004), we can define a function

$$Q(\mathbf{x}|\hat{\mathbf{x}}^{(t)}) = Q_1(\mathbf{x}|\hat{\mathbf{x}}^{(t)}) + Q_2(\mathbf{x}|\hat{\mathbf{x}}^{(t)}) \quad \text{such that} \quad (3.10)$$

$$L(\mathbf{x}) \leq Q_1(\mathbf{x}|\hat{\mathbf{x}}^{(t)}) + Q_2(\mathbf{x}|\hat{\mathbf{x}}^{(t)}), \quad (3.11)$$

verifying

$$L(\hat{\mathbf{x}}^{(t)}) = Q_1(\hat{\mathbf{x}}^{(t)}|\hat{\mathbf{x}}^{(t)}) + Q_2(\hat{\mathbf{x}}^{(t)}|\hat{\mathbf{x}}^{(t)}). \quad (3.12)$$

Due to (3.4), and assuming q a fixed value in the interval $[1, 2]$, $L_1(\mathbf{x})$ is an even and growing function with $|\mathbf{y} - \phi \mathbf{x}|$ (see figure (1(a))), having a slower growth, or equal case $q = 2$, than a quadratic function of $|\mathbf{y} - \phi \mathbf{x}|$. So, it makes sense to use as majorizer function of $L_1(\mathbf{x})$ a quadratic function, which is also an even function, so without a linear term. In other words, we can use as $L_1(\mathbf{x})$ majorizer a function of the form $\frac{1}{2} \sum_i a_i (\mathbf{y} - \phi \mathbf{x})_i^2 + b_i$. The upper bound function can be used in a MM algorithm only if it verifies the key property, i. e., the upper bound function must touch the function at the previous estimative. So it is necessary to determine a_i and b_i in order to

$$a_i (\mathbf{y} - \phi \mathbf{x})_i^2 + b_i \geq |(\mathbf{y} - \phi \mathbf{x})_i|^q, \quad \forall \mathbf{x} \quad (3.13)$$

$$a_i (\mathbf{y} - \phi \mathbf{x}')_i^2 + b_i = |(\mathbf{y} - \phi \mathbf{x}')_i|^q. \quad (3.14)$$

Let's consider the function $f(z, q)$ given by (3.3), and a quadratic majorizer such that

$$\begin{aligned} g_1(z, z') &= a z^2 + b \quad \text{such that:} \\ g_1(z, z') &\geq f(z, q) \quad \forall z \quad \text{and} \quad g_1(z', z') = f(z', q). \end{aligned} \quad (3.15)$$

Given that $f(z, q) = |z|^q$ we have, for $q \in [1, 2]$ and $z \neq 0$,

$$\frac{df(z, q)}{dz} = q|z|^{(q-1)} \text{sign}(z). \quad (3.16)$$

In order to $g_1(z, z')$ be tangent to $f(z, q)$ at $z = z'$, we get

$$a = \frac{q}{2}|z'|^{(q-2)}. \quad (3.17)$$

Since we also want $f(z', q) = g_1(z', z')$, we have

$$b = \frac{2-q}{2}|z'|^q. \quad (3.18)$$

Finally, we can write the majorizer function

$$g_1(z, z') = \frac{q}{2}(z')^{(q-2)}z^2 + \frac{2-q}{2}|z'|^q. \quad (3.19)$$

Based on majorizer (3.19) we can write

$$\begin{aligned} \frac{1}{2}\|\mathbf{y} - \phi \mathbf{x}\|_q^q &= \frac{1}{2} \sum_i |(\mathbf{y} - \phi \mathbf{x})_i|^q \\ &\leq \frac{q}{4} \sum_i |(\mathbf{y} - \phi \mathbf{x}')_i|^{(q-2)} (\mathbf{y} - \phi \mathbf{x})_i^2 \\ &\quad + \frac{2-q}{2} |(\mathbf{y} - \phi \mathbf{x}')_i|^q. \end{aligned} \quad (3.20)$$

Defining

$$\alpha_i^{(t)} = \frac{q}{2} \left(y_i - \sum_j (\phi_{ij} \hat{x}_j^{(t)}) \right)^{q-2}, \quad (3.21)$$

and

$$\beta_i^{(t)} = \frac{2-q}{2} \left| y_i - \sum_j (\phi_{ij} \hat{x}_j^{(t)}) \right|^q, \quad (3.22)$$

we have the majorizer $Q_1(\mathbf{x}|\hat{\mathbf{x}}^{(t)})$ given by

$$Q_1(\mathbf{x}|\hat{\mathbf{x}}^{(t)}) = \frac{1}{2} \sum_i \left[\alpha_i^{(t)} \left(y_i - \sum_j \phi_{ij} x_j \right)^2 + \beta_i^{(t)} \right]. \quad (3.23)$$

Since $L(\mathbf{x})$ is a function consisting of two separable terms, for which can be defined different majorizer functions, the majorizer function for the term $L_2(\mathbf{x}) = \lambda \|\mathbf{x}\|_p^p$ does not necessarily have to be a quadratic one. In fact, what is desirable is that the majorizer to be adopt for the $L_2(\mathbf{x})$ function, when added to the majorizer defined for $L_1(\mathbf{x})$, leads to a function $Q(\mathbf{x}, \hat{\mathbf{x}}^{(t)})$ with a minimizer easy to find. So, a ℓ_1 majorizer is the natural choice for penalties ℓ_p , with $0 < p \leq 1$, since it is more tighter than a quadratic majorizer. Recalling that, for the upper bound function can be used in a MM algorithm it must touch the function at the previous estimate, it is necessary to determine the parameters c and d so that

$$|x|^p \leq c|x| + d, \forall_x \text{ and } |x'|^p \leq c|x'| + d.$$

Let's consider the function $f(x, p)$ given by (3.3), and a majorizer ℓ_1 :

$$\begin{aligned} g_2(x, x') &= c|x| + d \text{ such that:} \\ g_2(x, x') &\geq f(x, p), \forall_x \text{ and } g_2(x', x') = f(x', p). \end{aligned} \quad (3.24)$$

Given that, for $p \in]0, 1]$, and $x' \neq 0$

$$\frac{df(x, p)}{dx} \Big|_{x=x'} = \text{sign}(x')p|x'|^{(p-1)}, \quad (3.25)$$

we have

$$c = p|x'|^{(p-1)}, \quad (3.26)$$

and

$$d = (1 - p)|x'|^p. \quad (3.27)$$

We can then write

$$g_2(x, x') = p|x'|^{(p-1)}|x| + (1 - p)|x'|^p. \quad (3.28)$$

Defining

$$\delta_i^{(t)} = \begin{cases} p|\hat{x}_i^{(t)}|^{(p-1)}, & \text{if } \hat{x}_i^{(t)} \neq 0 \\ 1/eps^a, & \text{if } \hat{x}_i^{(t)} = 0 \end{cases} \quad (3.29)$$

and

$$\epsilon_i^{(t)} = (1 - p)|\hat{x}_i^{(t)}|^p. \quad (3.30)$$

we have that the function $Q_2(\mathbf{x}|\hat{\mathbf{x}}^{(t)})$ can be written as

$$Q_2(\mathbf{x}|\hat{\mathbf{x}}^{(t)}) = \lambda \sum_i \left[\delta_i^{(t)} |x_i| + \epsilon_i^{(t)} \right]. \quad (3.31)$$

Given (3.10) we have that

$$\begin{aligned} Q(\mathbf{x}|\hat{\mathbf{x}}^{(t)}) &= \frac{1}{2} \sum_{i=1}^k \left[\alpha_i^{(t)} \left(y_i - \sum_j \phi_{ij} x_j \right)^2 + \beta_i^{(t)} \right] \\ &+ \lambda \sum_{i=1}^n \left[\delta_i^{(t)} |x_i| + \epsilon_i^{(t)} \right]. \end{aligned} \quad (3.32)$$

^a eps is the distance from 1.0 to the next larger double precision number, that is eps with no arguments returns $2^{(-52)}$.

Since at each step of the MM algorithm, we perform a minimization in \mathbf{x} , the terms $\beta_i^{(t)}$ and $\epsilon_i^{(t)}$ can be ignored, so we have

$$\begin{aligned} Q(\mathbf{x}|\hat{\mathbf{x}}^{(t)}) &= \frac{1}{2} \sum_i \left[\alpha_i^{(t)} \left(y_i - \sum_j \phi_{ij} x_j \right)^2 \right] \\ &+ \lambda \sum_i \left[\delta_i^{(t)} |x_i| \right], \end{aligned} \quad (3.33)$$

or, in vectorial notation

$$Q(\mathbf{x}|\hat{\mathbf{x}}^{(t)}) = \frac{1}{2} (\mathbf{y} - \phi \mathbf{x})^T \Gamma^{(t)} (\mathbf{y} - \phi \mathbf{x}) + \lambda \mathbf{1}^T \Lambda^{(t)} |\mathbf{x}|, \quad (3.34)$$

with

$$\Gamma^{(t)} = \begin{bmatrix} \alpha_1^{(t)} & 0 & \dots & 0 \\ 0 & \alpha_2^{(t)} & \dots & 0 \\ \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & \alpha_k^{(t)} \end{bmatrix}$$

and

$$\Lambda^{(t)} = \begin{bmatrix} \delta_1^{(t)} & 0 & \dots & 0 \\ 0 & \delta_2^{(t)} & \dots & 0 \\ \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & \delta_n^{(t)} \end{bmatrix}. \quad (3.35)$$

and where $|\mathbf{x}| = [|x_1|, |x_2|, \dots, |x_n|]^T$.

The minimization of the function $L(\mathbf{x})$ is implemented, iteratively, as a succession of minimalizations of the function $Q(\mathbf{x}|\hat{\mathbf{x}}^{(t)})$.

$$\begin{aligned} \hat{\mathbf{x}}^{(t+1)} &= \arg \min_{\mathbf{x}} Q(\mathbf{x}|\hat{\mathbf{x}}^{(t)}) \\ \hat{\mathbf{x}}^{(t+1)} &= \arg \min_{\mathbf{x}} \frac{1}{2} (\mathbf{y} - \phi \mathbf{x})^T \Gamma^{(t)} (\mathbf{y} - \phi \mathbf{x}) + \lambda \mathbf{1}^T \Lambda^{(t)} |\mathbf{x}|. \end{aligned} \quad (3.36)$$

Greater detail in the deduction of the presented mathematical expressions can be found in (Jardim, 2008).

4. Majorization-Minimization Algorithm

The minimization of the function $Q(\mathbf{x}|\hat{\mathbf{x}}^{(t)})$ (3.33) reflects an unconstrained convex optimization problem. In order to solve this problem with the algorithm GPSR-BB (Figueiredo *et al.*, 2007b),

it is necessary to reformulate the optimization problem (3.36) as a quadratic program (Nocedal & Wright, 1999), which leads to

$$\min_{\mathbf{z}} \mathbf{c}^T \mathbf{z} + \frac{1}{2} \mathbf{z}^T \mathbf{B} \mathbf{z} \quad (4.1)$$

subject to: $\mathbf{z} \geq 0$,

where $\mathbf{z} = [\mathbf{u}^T, \mathbf{v}^T]^T$ is a vector of unknown variables, with $\mathbf{u} = \max \{0, \mathbf{x}\}$ and $\mathbf{v} = \max \{0, -\mathbf{x}\}$, $\mathbf{c} = \lambda \mathbf{1}_{2n} + [-\mathbf{b}^T, \mathbf{b}^T]$ with $\mathbf{b} = \mathbf{A}^T \mathbf{y}$ and $\mathbf{A} = \phi^T \Gamma$, and $\mathbf{B} = \begin{bmatrix} \mathbf{A}^T \mathbf{A} & -\mathbf{A}^T \mathbf{A} \\ -\mathbf{A}^T \mathbf{A} & \mathbf{A}^T \mathbf{A} \end{bmatrix}$.

Considering the previously stated, the following algorithm was implemented in order to solve the optimization problem (4.1).

Step 0 (initialization): Given an initial estimate $\mathbf{z}^{(0)}$, set $t = 0$.

Step 1: Compute $\Gamma^{(t)}$ and $\tau^{(t)}$ according to (3.35) and $\tau_i = \lambda \Lambda_{ii}^{(t)}$, $\forall i=1, \dots, n$, respectively.

Step 2: Execute GPSR-BB algorithm with entries the current estimate $\hat{\mathbf{x}}^{(t)}$, the $\tau^{(t)}$ vector, $\mathbf{A} = \Delta^{(t)} \phi$, and $\mathbf{y}_m = \Delta^{(t)} \mathbf{y}$.

$$\hat{\mathbf{x}}^{(t+1)} = \text{GPSR-BB}(\mathbf{y}_m, \mathbf{A}, \tau^{(t)}, \hat{\mathbf{x}}^{(t)}).$$

Step 3: Perform convergence test and terminate with approximated solution $\hat{\mathbf{x}}^{(t+1)}$; otherwise set $t = t + 1$ and return to **Step 1**.

4.1. Stopping Criterion

Initially we used the general stopping criterion

$$\frac{\|\hat{\mathbf{x}}^{(t+1)} - \hat{\mathbf{x}}^{(t)}\|_2}{\|\hat{\mathbf{x}}^{(t)}\|_2} \leq \varepsilon, \quad (4.2)$$

and it was found that it leads to good results. After that we choose a stopping criterion more directed to the problem to be solved, adopting the one it was used in the GPSR-BB algorithm, where the algorithm stops when the relative change in the number of nonzero components of the estimate falls below a given bound value.

4.2. Computational cost analysis

The number of iterations required to find an approximate solution, both in the outer cycle as in the inner one (**Step 2**), is not possible to accurately predict, since it depends (among other factors) on the quality of the initial estimate $\hat{\mathbf{x}}^{(0)}$. However, is possible to analyze the computational cost of each iteration of the proposed algorithm. For outer cycle, each iteration computational cost is essentially the inherent in the calculation of the matrix $\Gamma^{(t)}$, vectors $\tau^{(t)}$ and \mathbf{y}_m . The computation of the matrix $\Gamma^{(t)}$ matrix, as well as the vector \mathbf{y}_m , implies a matrix-vector product. To compute $\Gamma^{(t)}$ it is necessary to multiply the $(k \times n)$ matrix ϕ , by the vector of estimates $\hat{\mathbf{x}}^{(t)}$, of dimension n . This operation has a cost of $O(kn)$. To compute the vector \mathbf{y}_m is necessary to multiply the matrix $\Gamma^{(t)}$, of dimension $(k \times k)$, by the vector \mathbf{y} , of dimension k , which has a computational cost of $O(k)$. Calculate the vector $\tau^{(t)}$ implies vector-scalar product, which requires n floating-point operations.

The main computational cost per each cycle iteration of the GPSR-BB algorithm is a small number of inner products, scalar-vector multiplications, and vectors additions, each one of them requiring n or $2n$ floating-point operations, , plus a modest number of multiplications by \mathbf{A} and \mathbf{A}^T . The algorithm proposed in this paper for the resolution of the optimization problem that results from the minimization of the function $L(\mathbf{x})$ (2.1), executes GPSR-BB algorithm, where the matrix \mathbf{A} results from the product of the matrices ϕ and $\Gamma^{(t)}$. Given that ϕ is a $k \times n$ matrix, the computational cost of direct implementation of matrix-vector products by ϕ or ϕ^T is $O(kn)$. For $\Gamma^{(t)}$, $k \times k$ matrix, the computational cost of direct implementation of matrix-vector products is $O(k)$. If $\phi = \mathbf{R}\mathbf{W}$ is a matrix of dimension $k \times n$ and \mathbf{R} a $k \times d$ matrix, then \mathbf{W} must be a $d \times n$ matrix. If \mathbf{W} contains an orthogonal wavelet basis ($d = n$), matrix-vector products involving \mathbf{W} or \mathbf{W}^T can be implemented using fast wavelet tranform algorithms with $O(n)$ cost (Mallat, 1999), instead of the $O(n^2)$ cost of a direct matrix-vector product. Consequently, the cost of a product by ϕ or ϕ^T is $O(n)$ plus that of multiplying by \mathbf{R} or \mathbf{R}^T which, with a direct implementation, is $O(kn)$.

4.3. Convergence analysis

In order to analyze the convergence of the algorithm proposed in this paper, first will be analyzed the convergence of the GPSR-BB algorithm used in each iteration of the majorization-minimization algorithm, whose entries differ from those of the original algorithm of Figueiredo, being given by the equations defined above. Secondly will be analyzed the convergence of the iterative algorithm defined by the update (3.36).

As stated by Figueiredo in (Figueiredo *et al.*, 2007b) the convergence of the algorithm GPSR-BB used in this work can be derived from the analysis of Bertsekas (Bertsekas, 1999), but follows most directly from the results of Serafini, Zanghirati, and Zanni (Serafini *et al.*, 2005). In the algorithm proposed in this work we use the GPSR-BB algorithm with entries different from the ones defined by Figueiredo (Figueiredo *et al.*, 2007b). To summarize convergence properties of the GPSR-BB algorithm with the entries previously defined, we assume that termination occurs only when $\mathbf{x}^{(t+1)} = \mathbf{x}^{(t)}$, which indicates that $\mathbf{x}^{(t)}$ is optimal.

Theorem 1: When $p = 1 \wedge q \in [1, 2]$ the sequence of iterates generated by the GPSR-BB algorithm with the entries \mathbf{y}_m , \mathbf{A} , $\tau^{(t)}$ and the current estimate $\hat{\mathbf{x}}^{(t)}$ either terminates at a solution of (4.1) or else converges to a solution of (4.1) at an R-linear rate.

Proof. Theorem 2.1 of (Serafini *et al.*, 2005) can be used to demonstrate that all accumulation points of $\{\mathbf{x}^{(t)}\}$ are stationary points. Although, in (Serafini *et al.*, 2005), this result applies to an algorithm in which the steplength parameters $\alpha^{(t)}$ used in the gradient projection method are chosen by a different scheme, the only relevant requirement on these parameters in the proof of [(Serafini *et al.*, 2005), Theorem 2.1] is that they lie in the range $[\alpha_{\min}, \alpha_{\max}]$, as in the case of the GPSR-BB algorithm used in this work. When $p = 1 \wedge q \in [1, 2]$, the objective in (4.1) is convex and bounded below, we can apply [(Serafini *et al.*, 2005), Theorem 2.2] to deduce convergence to a solution of (4.1) at an R-linear rate. When $p < 1$, the objective function in (4.1) is nonconvex; thus it can not be guaranteed that the algorithm converges to a global optimum. Nevertheless, in practice, we have never observed any convergence problems: the results show that the proposed algorithm finds actually a minimum, which although may not be a global minimum corresponds to a good reconstruction of the signal. \square

Theorem 2: For $Q(\mathbf{x}, \mathbf{x}')$ a continuous function in $(\mathbf{x}, \mathbf{x}')$ and L a strictly convex function, the MM iteration sequence $\hat{\mathbf{x}}^{(t)}$ converges to the global minimum of L .

Proof. Knowing that $\hat{\mathbf{x}}^{(t+1)} = \arg \min_{\mathbf{x}} Q(\mathbf{x}|\hat{\mathbf{x}}^{(t)})$, and recalling that the majorizer function Q verifies the conditions given by (3.6) and (3.7), we have

$$L(\hat{\mathbf{x}}^{(t+1)}) \leq Q(\hat{\mathbf{x}}^{(t+1)}|\hat{\mathbf{x}}^{(t)}) \leq Q(\hat{\mathbf{x}}^{(t)}|\hat{\mathbf{x}}^{(t)}) = L(\hat{\mathbf{x}}^{(t)}), \quad (4.3)$$

where the left hand inequality follows from the definition of Q and the right hand inequality from the definition of $\hat{\mathbf{x}}^{(t+1)}$. The sequence $L(\hat{\mathbf{x}}^{(t)})$, for $t = 1, 2, \dots$, is, therefore, nonincreasing. Under mild conditions, namely that $Q(\mathbf{x}, \mathbf{x}')$ is continuous in $(\mathbf{x}, \mathbf{x}')$, all limit points of the MM sequence $L(\hat{\mathbf{x}}^{(t)})$ are stationary points of L , and $L(\hat{\mathbf{x}}^{(t)})$ converges monotonically to $L^* = L(\mathbf{x}^*)$, for some stationary point \mathbf{x} . If, additionally, L is strictly convex, $\hat{\mathbf{x}}^{(t)}$ converges to the global minimum of L . Proofs of these properties are similar to those of similar properties of the EM algorithm (Huber & Ronchetti, 2009). \square

5. Experiments

In this section are presented, analyzed and discussed the results obtained by the proposed algorithm in compressed sensing applications and in the reconstruction of sparse images or with sparse representations, where for each signal the algorithm is tested for different values of \mathbf{p} and \mathbf{q} . Parameter λ is hand-tuned for the best SNR improvement. For compressed sensing scenarios it is adjusted according to the expression $\lambda = 0.1 \|\phi^T \mathbf{y}\|_{\infty}$ (as suggested by Fuchs in (Fuchs, 2004)).

All the experiments reported in this section were obtained with MATLAB (MATLAB 7.0 R14) implementations of the algorithm described above. The computing platform is a standard personal computer with Intel(R) Core(TM) i7 CPU, 8 GB of RAM, and running Windows 7 operating system.

5.1. Compressed Sensing

We first consider a typical compressed sensing (CS) scenario, where the goal is to reconstruct a length- n sparse signal (in the canonical basis, thus $\mathbf{W} = \mathbf{I}$ and $d = n$) from k observations, where

$k \leq n$. The rows of the $k \times n$ observation matrix \mathbf{R} are unit-norm random vectors (of Gaussian components) in \mathcal{R}^n . Notice that, since $k \leq n$, the system $\mathbf{R}\mathbf{x} = \mathbf{y}$ is undetermined. In the first example, we take $n = 2^{11} = 2048$, $k = 2^8 = 256$, and generate \mathbf{y} by adding Laplacian noise (probability density function $f(x) = a e^{-\frac{|x|}{a}}$) with parameter $a = 0.01$ to $\mathbf{R}\mathbf{x}$ (figure (2(b))). The original sparse signal \mathbf{x} is generated by a mixture of a uniform distribution on $[1, 1]$ and a point mass at zero, with probabilities 0.01 and 0.99, respectively. As we can see in 2(a), the original signal is indeed sparse.

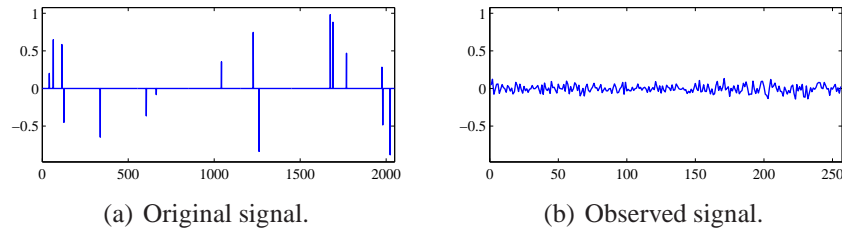


Figure 2. Original and observed signal.

For the described data set we have as initial estimate for $\ell_q - \ell_p$ algorithm the signal $\hat{\mathbf{x}}^{(0)} = \phi^T \mathbf{y}$. The estimates obtained by solving (2.1) using the proposed algorithm for $q = 1 \wedge p \in]0, 1]$ are shown in figure (3), and it can be verified that they are very close to the original signal.

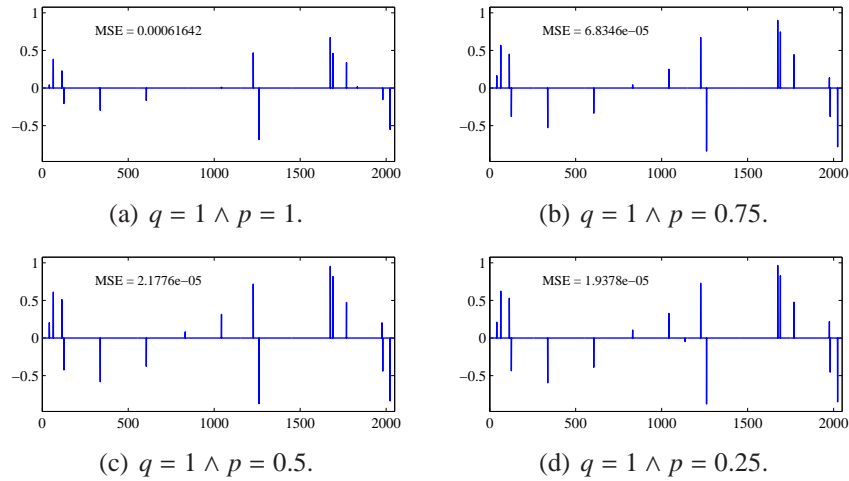


Figure 3. Estimated signal by (2.1) minimization for $q = 1 \wedge p \in]0, 1]$

From the results obtained, and presented in figure (3) it is possible to observe that the approximation mean squared error^b (MSE) for $q = 1 \wedge p = 1$ is about 10 times greater than that obtained for $p \leq 1$, meeting the expected results.

^bMSE = $(1/n) \|\hat{\mathbf{x}} - \mathbf{x}\|_2^2$, where $\hat{\mathbf{x}}$ is an estimate of \mathbf{x} .

As referred above, the generalization of the data term from a ℓ_2 norm to a ℓ_q norm, gives to the approximation criterion statistical strength features when $q < 2$. This can be verified in the second experiment, where we take $n = 2^{11} = 2048$, $k = 2^8 = 256$, and generate y by adding to $\mathbf{R}\mathbf{x}$ impulsive noise. Considering the original signal represented in figure (4(a)) with 16 nonzero components, we can see in figure (5) that the MSE of the approximation decreases with the value of q , taken as constant the value of $p = 1$.

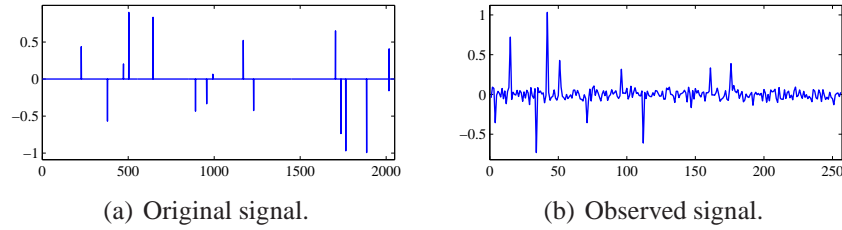


Figure 4. Original and observed signal.

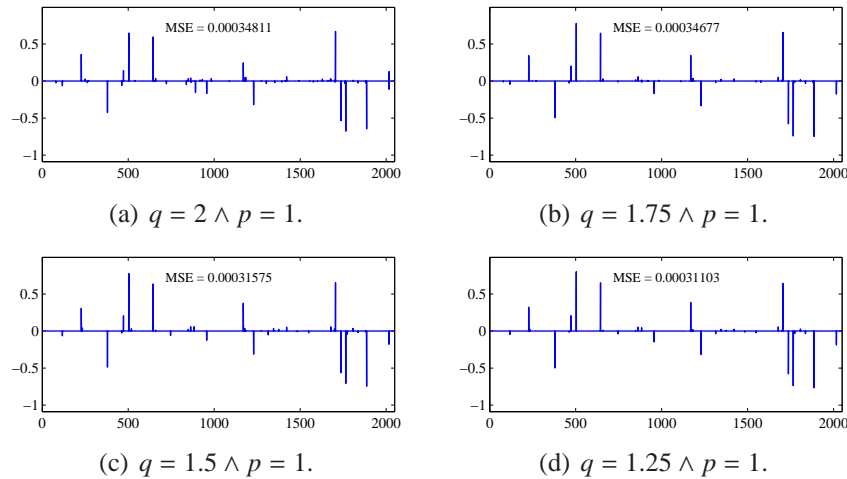


Figure 5. Estimated signal by (2.1) minimization for $q \in]1, 2] \wedge p = 1$

Note, for example, that the best result achieved by $\ell_q - \ell_p$ algorithm for $q = 2$ ($\text{MSE} = 3.4811 \times 10^{-4}$, 59 nonzero components) is not so good as the obtained for $q = 1.25$ ($\text{MSE} = 3.1103 \times 10^{-4}$, 35 nonzero components). Although a significant improvement in quantitative terms (the values of the approximations MSE for different values of q are not too different) doesn't occur, in this case, we can verify that, as the value of q decreases, the approach presents qualitative improvements (lower number of spurious observations in the final estimate). Figures (6(a)) and (6(b)) show the evolution of the MSE and objective function, for outer and inner loop respectively, against iteration number of the proposed algorithm, when $q = 1.5 \wedge p = 1$.

Analyzing the graphics of figure (6) we can observe that the values of the approximation MSE and the objective function decreases in each iteration of the $\ell_q - \ell_p$ algorithm. In fact, starting

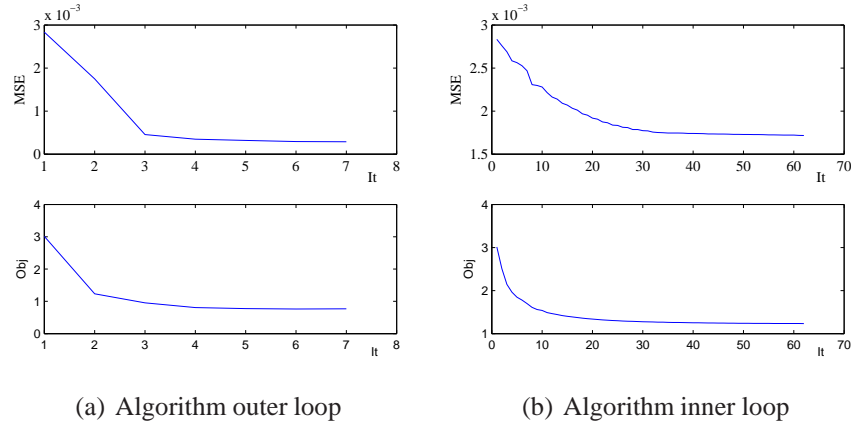


Figure 6. MSE and Objective function vs. Iteration number

from a initial upper bound function, built on the initial estimate, it is observed that the proposed algorithm builds at each iteration a new one, on the basis of the obtained solution for the previous upper bound function, which solution is closest to the original signal.

Next experiment shows the performance of the implemented algorithm in a typical compressed sensing problem, where the goal is to reconstruct a signal from k projections (with $k = 2^{10} = 1024$). The observation matrix \mathbf{R} , of dimension $k \times n$, is a matrix of random projection vectors. The two-dimensional original signal, which is represented in figure (7) has a sparse wavelet transform, and in this example the columns of the representation matrix \mathbf{W} form an orthogonal wavelet basis (daubechies-2 (Haar)). The original signal is an image of piecewise smooth filtered white noise of dimension $n' \times n'$, with $n' = 2^6$, *i. e.* $n = 2^{12}$ (figure (7)). The observed signal is obtained by adding white Gaussian noise with standard deviation 0.001 to $\mathbf{R}\mathbf{x}$. Figures (8(a)) - (8(c)) shows the results of $\ell_q - \ell_p$ algorithm, taken as initial estimate $\hat{\mathbf{x}}^{(0)} = 0$. By the results we can see that the proposed algorithm produces, from k projections corrupted by random white Gaussian noise best approximations (lower MSE) for $p < 1$.

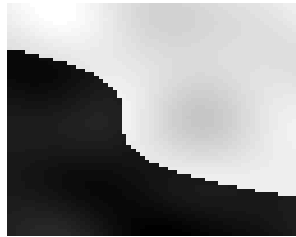


Figure 7. Original signal.

5.2. Image Restoration

The following two experiments show the performance of the proposed algorithm in real images restoration, where the columns of representation matrix \mathbf{W} form an orthogonal wavelet basis

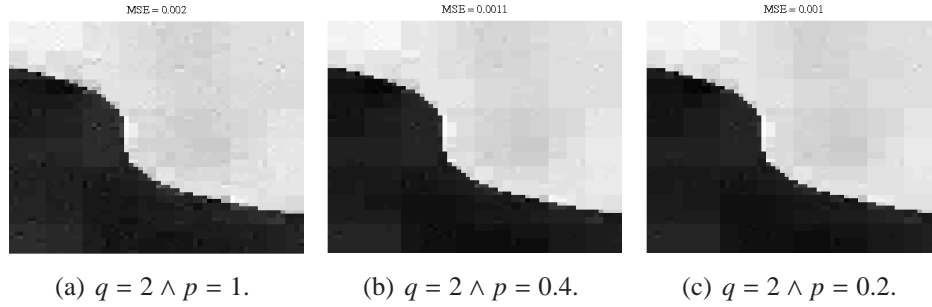


Figure 8. Estimated signal by (2.1) minimization, for $q = 2 \wedge p \in]0, 1]$.

(daubechies-2 (Haar)). The observation matrix \mathbf{R} , of dimension $k \times k$, is a Toeplitz blocks matrix representing 2D convolutions. In the first of these experiments we have as original signal the well-known Camera-man image (figure (9(a))). The observed image is obtained convolving the original image with a uniform blur filter of size 9×9 , and then adding to the blurred image ($\mathbf{R}\mathbf{x}$) white Gaussian noise with standard deviation 0.005 (figure (9(b))). Taken as initial estimate $\hat{\mathbf{x}}^{(0)} = \phi^T \mathbf{y}$, figures (10(a)) - (10(d)) shows the algorithm performance for $q = 2$ and $p \in]0, 1]$. For $q = 2 \wedge p = 0.5$ the initial value of the objective function is 2.0834×10^6 and the final one is 3.146×10^4 .

A typical statistical model for image wavelet coefficients is the Generalized Gaussian Density (Moulin & Liu, 1999), given by $p(\theta) = \exp \left\{ -\frac{|\theta|^p}{\alpha} \right\}$, where θ represents the wavelet coefficients vector. The graph shown in figure (11) represents the evolution of MSE with p , where we can see that the best approximation occurs for $p = 0.5$. This is consistent with Moulin's statement (Moulin & Liu, 1999) that good image models based on wavelets are obtained doing $p \simeq 0.7$.

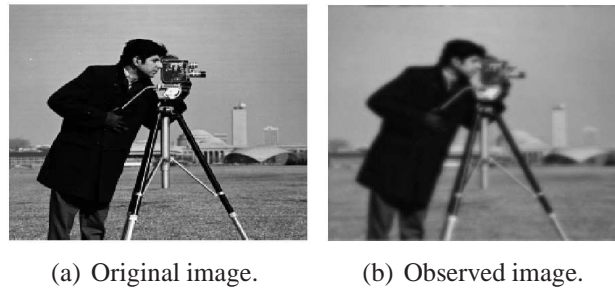


Figure 9. Original and observed real image.

In the last reported experiment the original image is the also well-known Lenna image (figure (12(a))). The observed image is obtained convolving the original image with a uniform blur filter of size 9×9 , and then adding to the blurred image ($\mathbf{R}\mathbf{x}$) impulsive noise (figure (12(b))).

Figures (13(a)) - (13(d)) shows the performance of the implemented algorithm for $q \in]1, 2]$ and $p = 1$. Again we can observe that best results are achieved for values of p below 1. For this experiment, and for $q = 1.25 \wedge p = 1$, we have that the initial value of the objective function is 3.7745×10^6 and the final one is 1.689×10^5 .

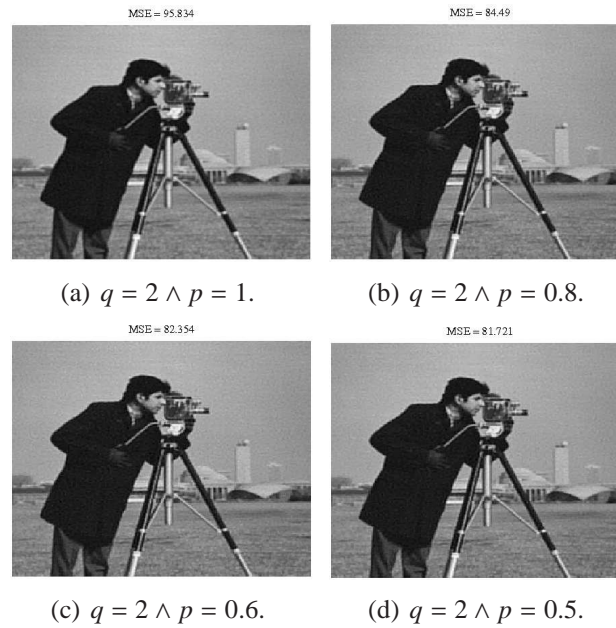


Figure 10. Estimated signal by (2.1) minimization, for $q = 2 \wedge p \in]0, 1[$.

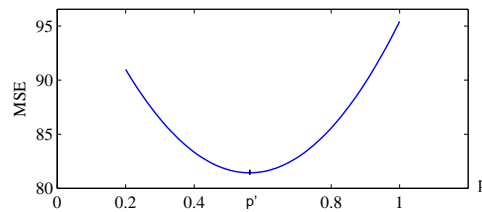


Figure 11. Approximation MSE evolution with p .

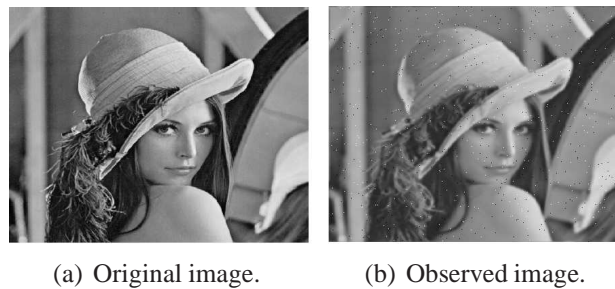


Figure 12. Original and observed real image.

6. Conclusions

In this paper was proposed a majorization-minimization class algorithm to address ℓ_q - ℓ_p optimization problems, where $p \in]0, 1[\wedge q \in [1, 2]$, of which ℓ_2 - ℓ_1 is an instance. The proposed algorithm was tested on scenarios of compressed sensing and image reconstruction, and, in both



Figure 13. Estimated signal by (2.1) minimization, for $q \in]1, 2]$ and $p = 1$.

cases, experiments were performed for data corrupted by uniform and impulsive noise. As mentioned, the generalization of the coefficients term to be estimated, from a ℓ_1 to a ℓ_p norm, with $p < 1$, aims to move towards the solution of the original combinatorial optimization problem, whose complexity prevents the calculation of a global solution in polynomial time. Although no one can guarantee the convergence of the algorithm to a global minimum, whereas for $p < 1$ the problem is no longer convex, the analysis of the results shows that the proposed algorithm provides as solution of the optimization problem a minimum corresponding to a signal reconstruction better than the one obtained by making $p = 1$ (lowest approximation MSE).

While in the experiments with nonnatural sparse signals we verify that approximation MSE, relatively to the original signal, decreases with the value of p , the same is not true for natural images, where we verify that there is an optimal value of $p < 1$, for which we obtain the best possible approximation, *i.e.*, that which corresponds the smallest MSE. This is justified against the model used for wavelet coefficients (GGD - Generalized Gaussian Density), which is function of the parameter p .

In the presence of outliers the proposed algorithm was tested taking p constant and equal to 1, and ranging q in $]1, 2]$. Both in compressed sensing applications of unidimensional signals, as in natural images restoration applications we can verify that the approximation MSE decreases with the value of parameter q . From the analysis of the results we can still observe that, as the value of q decreases the amount of outliers of the optimization problem solution also decreases. Hence it can be concluded that the use of ℓ_q norms, with $q < 2$, in data term, gives to the approximation criterion statistical robustness characteristics.

References

- Bertsekas, D. P. (1999). *Nonlinear Programming*. 2nd edition ed.. Athena Scientific.
- Candes, E. and T. Tao (2005). Decoding by linear programming. *IEEE Transactions on Information Theory* **51**(12), 4203–4215.
- Casas, E., C. Clason and K. Kunisch (2012). Approximation of elliptic control problems in measure spaces with sparse solutions. *SIAM Journal on Control and Optimization* **50**(4), 1735–1752.
- Chen, S., D. L. Donoho and M. A. Saunders (2001). Atomic decomposition by basis pursuit. *SIAM Journal on Scientific and Statistical Computing, Review* **43**, 129–159.
- Claerbout, J. and F. Muir (1973). Robust modelling of erratic data. *Geophysics* **38**, 826–844.
- Davis, G., S. Mallat and M. Avellaneda (1997). Adaptive greedy approximation. *Journal Constructive Approximations* **13**(1), 57–98.
- Donoho, D. L. (2006). Compressed sensing. *IEEE Transactions on Information Theory* **52**, 1289–1306.
- Efron, B., T. Hastie, I. Johnstone and R. Tibshirani (2004). Least angle regression. *The Annals of Statistics* **32**(2), 407–499.
- Figueiredo, M. A. T. and R. Nowak (2003). An em algorithm for wavelet-based image restoration. *IEEE Transactions on Image Processing* **12**, 906–916.
- Figueiredo, M. and R. Nowak (2005). A bound optimization approach to wavelet-based image deconvolution. In: *IEEE International Conference on Image Processing - ICIP'2005*.
- Figueiredo, M., J. Bioucas-Dias and R. Nowak (2007a). Majorization-minimization algorithms for wavelet-based image restoration. *IEEE Transactions on Image Processing: Special Issue on Convex Optimization Methods for Signal Processing* **16**(12), 2992–3004.
- Figueiredo, M., R. Nowak and S. Wright (2007b). Gradient projection for sparse reconstruction: Application to compressed sensing and other inverse problems. *IEEE Journal of Selected Topics in Signal Processing: Special Issue on Convex Optimization Methods for Signal Processing* **1**(4), 586–598.
- Fuchs, J.-J. (2004). On sparse representations in arbitrary bases. *IEEE Transactions on Information Theory* **50**, 1341–1344.
- Herzog, R., G. Stadler and G. Wachsmuth (2012). Directional sparsity in optimal control of partial differential equations. *SIAM Journal on Control and Optimization* **50**(2), 943–963.
- Huber, P. J. and E. M. Ronchetti (2009). *Robust Statistics*. 2nd edition ed.. Wiley.
- Hunter, D. and K. Lange (2004). A tutorial on mm algorithms. *The American Statistician* **58**, 30–37.
- Jardim, S. (2008). Algoritmos para Representacao Esparsa e Robusta de Sinais. PhD thesis. Instituto Superior Tecnico, Universidade Tecnica de Lisboa. In Portuguese.
- Malioutov, D., M. Cetin and A. Willsky (2005). Homotopy continuation for sparse signal representation. In: *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing*. Vol. 5. Philadelphia. pp. 733–736.
- Mallat, S. (1999). *A Wavelet Tour of Signal Processing*. 2nd edition ed.. Academic Press.
- Moulin, P. and J. Liu (1999). Analysis of multiresolution image denoising schemes using generalized-gaussian and complexity priors. *IEEE Transaction on Information Theory* **45**, 909–919.
- Natarajan, B. (1995). Sparse approximate solutions to linear systems. *SIAM Journal on Computing* **24**(2), 227–234.
- Nocedal, J. and S. Wright (1999). *Numerical Optimization*. Springer-Verlag. New York.
- Serafini, T., G. Zanghirati and L. Zanni (2005). Gradient projection methods for quadratic programs and applications in training support vector machines. *Optimization Methods and Software*.
- Stadler, G. (2009). Elliptic optimal control problems with l1-control cost and applications for the placement of control devices. *Computational Optimization and Applications* **44**(2), 159–181.

- Turlach, B. (2005). On algorithms for solving least squares problems under l_1 penalty or an l_1 constraint. In: *Proceedings of the American Statistical Association; Statistical Computing Section*. Alexandria. pp. 2572–2577.
- Turlach, B., W. N. Venables and S. J. Wright (2005). Simultaneous variable selection. *Technometrics* **27**, 349–363.
- Wright, S. J., R. D. Nowak and M. A. T. Figueiredo (2009). Sparse reconstruction by separable approximation. *IEEE Transactions on Signal Processing* **57**, 2479–2493.
- Zarzer, C. A. (2009). On tikhonov regularization with non-convex sparsity constraints. *Inverse Problems* **25:025006**, 13 pp.



Logical Tree of Mathematical Modeling

László Pokorádi*

Óbuda University Donát Bánki Faculty of Mechanical and Safety Engineering, 1081 Népszínház u. 8., Budapest, Hungary.

Abstract

During setting up a mathematical model, it can be very important and difficult task to choose input parameters that should be known for solution of this problem. A similar problem might come up when someone wants to carry out an engineering calculation task. A very essential aim technical education is developing of good logical engineering thinking. One main part of this thinking is to determine the potential sets of required input parameters of an engineering calculation. This paper proposes a logical tree based method to determine the required parameters of a mathematical model. The method gives a lively description about needed data base, and computational sequence for us to get to determine the set of required output parameter. The shown method is named **LogTreeMM - Logical Tree of Mathematical Modeling**.

Keywords: mathematical modeling, logical tree, engineering thinking, STEM education.
2010 MSC: 93A30, 00A71, 97D30.

1. Introduction

On the one hand, during engineering work, dozens and dozens of times we should determine any parameters of a technical system or process. To carry out the task mentioned above we have to know some input parameters of the investigated system or process. We can meet similar task during setting up a mathematical model for system or process simulation. The correct identification of optimal set of required input parameters is an important and hard engineering task.

On the other hand, during technical education it is a very difficult and essential task to develop good logical engineering thinking of students or pupils. One main part of this thinking is to determine the optimal set of required input parameters of the calculation task mentioned above.

The main aim of this paper is to show a logical tree method to determine required parameters of a mathematical model or an engineering calculation. This method gives a lively description about needed data base, and computational sequence for us to get to determine the required output

*Corresponding author

Email address: pokoradi.laszlo@bgk.uni-obuda.hu (László Pokorádi)

parameter. As the flow chart emphasizes the main steps of a calculation task, the proposed logical tree demonstrates the interdependencies and interrelations of variables. To choose a set of required parameters, firstly we should have an applicable equation to calculate the output parameter of the system that is a dependent variable of its model. Knowing this adaptable equation we can face the next question: How can we determine the independent variable(s) of the foregoing equation? And we should ask it repeatedly. ... It is possible that we can furnish two or more answers to one of these questions. In this case we get different required model parameter(s).

It is easily statable that we use logical inferences during determination of needed parameters to set up and to apply a mathematical model. In the one hand we use AND logical operations if all parameters should be known. On the other hand we use OR logical operations if we know two or more equations to calculate a parameter.

These logical connections are used in Fault Tree Analysis to determine causes of a system failure. The handbooks of NASA ([Stamatelatos & Caraballo, 2002](#)) and U.S. Nuclear Regulatory Commission ([Vesely et al., 1987](#)) show theoretical background and practical questions of FTA. There are several publications that propose reliability or safety methods based on FTA. For example, Tchorzewska-Cieslak and Boryczko presented the methodology of the FTA and an example of its application in order to analyze different failure scenarios in water distribution subsystem ([Tchorzewska-Cieslak & Boryczko, 2010](#)). They concluded that the FTA is particularly useful for the analysis of complex technical systems in which analysis of failure scenarios is a difficult process because it requires to examine a high number of cause-effect relationship. The water distribution subsystem undoubtedly belongs to such systems. The FTA involves thinking back, which allows the identification of failure events that cause the occurrence of the Top Event. In the case of very large fault trees it is advisable to use computer methods ([Tchorzewska-Cieslak & Boryczko, 2010](#)).

One of the FTA-based methods is the so-called bow-tie model that was used by Markowski and Kotynia ([Markowski & Kotynia, 2011](#)). It consists of a Fault Tree (FT) which identifies the causes of the undesired top event, and an Event Tree (ET) showing what is the consequence of such a release. So, this method encompasses the complete accident scenario using a bow-tie created by a Fault Tree and an Event Tree.

Pokorádi showed the adaptation of linear mathematical diagnostic modeling methodology for setting-up of Linear Fault Tree Sensitivity Model (LFTSM) ([Pokorádi, 2011](#)). The LFTSM is a modular approach tool that uses matrix-algebraic method based upon the mathematical diagnostic methodology of aircraft systems and gas turbine engines.

The logical method being presented in this paper is an adaptation of logical construction part of Fault Tree Analysis (FTA). This proposed method is named **LogTreeMM - Logical Tree of Mathematical Modeling**.

Pokorádi and Molnár showed the methodology of the Monte-Carlo Simulation and its applicability to investigate influences of fluid parameters to system losses by an easy pipeline system model. The (basically theoretical) obtained consequents and experiences can be used for investigation of parametrical uncertainties of the geothermal pipeline system, such as fluid characteristics indeterminations ([Pokorádi & Molnár, 2011](#)). This simulation model is practically used for demonstrating methodology of the proposed method.

The rest of this paper is organized as follows: Section 2 recalls the FTA methodology shortly

and the logical tree method theoretically. Section 3 presents a possibility of use of the proposed method by a case study. Section 4 summarizes the paper, outlines the prospective scientific work of the Author.

2. Theoretical delineation

The proposed method is an adaptation of logical construction of FTA. The FTA is a systematic, deductive (top-down type) and probabilistic risk assessment tool, which shows the causal relations leading to a given undesired event. Bell Telephone Laboratories developed its concept at the beginning of the 1960s. It was adopted later and extensively applied by Boeing Company. FTA is one of several symbolic "analytical logic techniques" found in operations research and in system reliability. The FTA is particularly useful for the analysis of complex technical systems where analysis of failure scenarios is a difficult process because it requires the examination of a high number of cause-effect relationships.

Fault Tree diagram displays on undesired state of the investigated system (Top Event) in terms of the states of its components (Basic Events). The FTA is a graphical design technique main result of which is a graph that has a dendritic structure.

The first step in a FTA is the selection of the Top Event that is a specific undesirable state or failure of a system.

After having analyzed the system so that we know all the causing effects we can construct the fault tree. Fault tree is based on **AND** and **OR** gates, which define the major characteristics of the fault tree.

The **AND** logical gate (Table 1) should be used if output event occurs only if all input events occur simultaneously. If the output event occurs if any of the input events occur, either alone or in any combinations, the **OR** logical gate (Table 1) should be used.

The Figure 1 shows a demonstrative Fault Tree. In the figure event B or C fail is Intermediate Event. The events A; B and C are Basic Events.

(After having the fault tree of the investigated undesired event, the probability of the Top Event can be analyzed depending of the probability of Basic Events. But it is not interesting in this study.)

The proposed method is analog with the above reviewed FTA technique. To construct LogTreeMM the following definitions are used:

A parameter can be known directly when

- its value is well-known (for example material characteristics);

or.

- it can be determined by direct measurement (for example internal diameter of a tube).

Let a variable be named Top Parameter if it should be determined but it is not used to calculate other one(s) in the investigated situation (is not an intermediate parameter).

Let a variable be called Intermediate Parameter if it has be known to calculate other one(s) but it cannot be known directly.

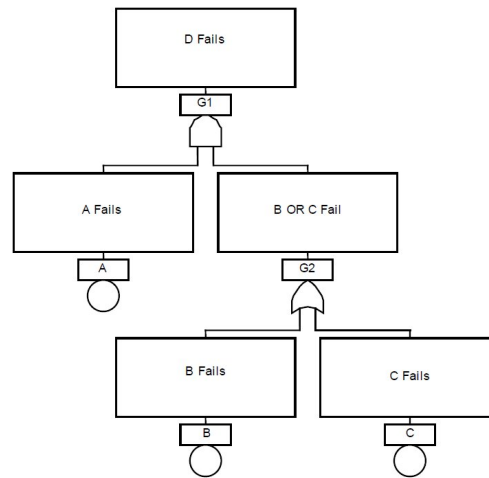







Figure 1. A Fault Tree (source: (Stamatelatos & Caraballo, 2002)).

Table 1		
Symbols and Analogies between FTA and LogTreeMM		
Symbol	FTA	LogTreeMM
	Top Event	Top Parameter
	Intermediate Event	Intermediate Parameter
	Basic Event	Basic Parameter
	AND logical gate	
	OR logical gate	

Let a variable be named Base Parameter if

- it is known directly;

or

- it cannot be determined by any equation (relation).

Let an **AND** logical gate (Table 1) be used if all of the independent variables should be known to calculate a dependent variable of the given equation (relation).

Let an **OR** logical gate (Table 1) be used if there are more than one equation (relation) on even terms to calculate the given dependent variable.

Table 1 demonstrates symbols used in FTA and LogTreeMM, as well as the analogies between their events and gates.

3. Case Study

To demonstrate step by step the logical tree method introduced above, we show a case study based on the simulation model presented in [3]. The study aimed the investigation of influences of

fluid parameters to system losses in case of an easy pipeline system model. (Further on let the i -th logical gate be labeled by $/i/$.)

The illustrative system consisted of one lineal pipe and only one pipe fitting. The Δp pressure loss of this pipeline system as the Top Parameter can be determined by the equation

$$\Delta p = \Delta p_{cs} + \Delta p_{sz} \quad (3.1)$$

where:

Δp_{cs} pressure loss of linear pipe;

Δp_{sz} pressure loss of pipe fitting.

Thus two parameters (Δp_{cs} **AND** Δp_{sz}) should be known, which is shown by the **AND** gate of Figure 2. However, we do not know them directly. They are symbolized by rectangles, because they are Intermediate Elements of our investigation.

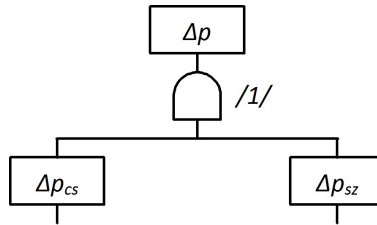


Figure 2. Logical Gate /1/ for Equation (3.1).

On the second level, the certain two structural elements (lineal pipe and pipe fitting) should be investigated. In the left branch, the pipe loss of linear pipe can be calculated by the equation

$$\Delta p_{cs} = \frac{\rho}{2} c^2 \frac{l}{d} \lambda \quad (3.2)$$

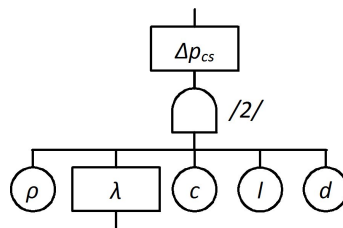


Figure 3. Logical Gate /2/ for Equation (3.2).

For that we should know fluid density ρ **AND** average fluid velocity c **AND** tube length l **AND** internal diameter d **AND** pipe loss coefficient λ . This logical sequence is shown by **AND** gate in Figure 3.

Except of the pipe lost coefficient (the rectangle shows that it is an Intermediate Parameter) all parameters can be determined directly (they are Basic Parameters), therefore they are shown by circles in Figure 3.

The pipe loss coefficient can be determined, depending only on Reynolds-number Re that cannot be determined directly (see Figure 4), by empirical equations in case of different Reynolds-number intervals:

$Re < 2320$:

$$\lambda = \frac{64}{Re} \quad (3.3)$$

$2320 < Re < 8 \cdot 10^4$:

$$\lambda = \frac{0.316}{\sqrt[4]{Re}} \quad (3.4)$$

$2 \cdot 10^4 < Re < 2 \cdot 10^6$:

$$\lambda = 0.0054 + 0.396 Re^{-0.3} \quad (3.5)$$

$10^5 < Re < 10^8$:

$$\lambda = 0.0032 + 0.211 Re^{-0.337} \quad (3.6)$$

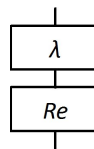


Figure 4. Connection of Equations (3.3) - (3.6).

For determination of the Reynolds-number Re the following equation should be used

$$Re = \frac{c d}{\nu} \quad (3.7)$$

For that we should know the average fluid velocity c **AND** the internal diameter d **AND** the kinematic viscosity of the fluid ν (they can be determined directly in other words, they are Basic Parameters).

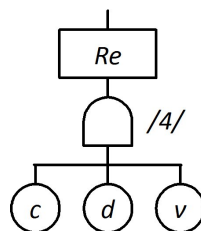


Figure 5. Logical Gate /4/ for Equation (3.7).

In the right branch of the second level, the loss pressure of pipe fitting can be determined by any of the following two equations

$$\Delta p_{sz} = \frac{\rho}{2} c^2 \xi \quad (3.8)$$

$$\Delta p_{sz} = \frac{\rho}{2} c^2 \frac{l_e}{d} \lambda \quad (3.9)$$

It is represented by the **OR** logical gate /3/ in Figure 6.

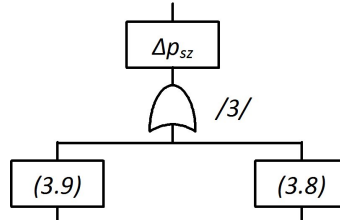


Figure 6. Logical Gate /3/.

In case of Equation (3.8), we need to know fluid density ρ **AND** average fluid velocity c **AND** pipe fitting loss coefficient ξ (see Figure 7).

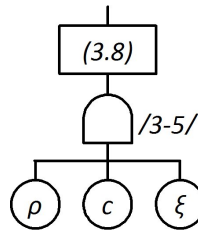


Figure 7. Logical Gate /3 – 5/ for Equation (3.8).

If the Equation (3.9) is used, the following parameters have to be known: fluid density ρ **AND** average fluid velocity c **AND** equivalent pipe length of pipe fitting l_e **AND** internal diameter of tube d **AND** pipe loss coefficient of tube λ (see Figure 8). The "equivalent pipe length" is length of given pipe, of which loss is equal to the loss of investigated pipe fitting in case of equal average fluid velocity.

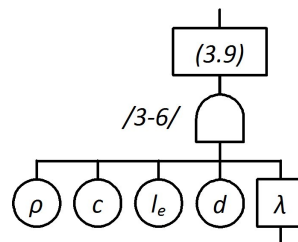


Figure 8. Logical Gate /3-6/ of Equation (3.9).

Assembling of the Figures 2 - 8, we get the logical tree of the illustrative model that is represented in Figure 9.

The sets of needed input parameters can be determined easily by investigating of the Logical Tree. For this purpose the subsets of the known parameters of the logical gates should be deducted first. In our case:

$$x_1 = \emptyset \quad (3.10)$$

$$x_2 = \{\rho; c; l; d\} \quad (3.11)$$

$$x_3 = \emptyset \quad (3.12)$$

$$x_4 = \{c; d; v\} \quad (3.13)$$

$$x_{3-5} = \{\rho; c; \xi\} \quad (3.14)$$

$$x_{3-6} = \{\rho; c; l_e; d\} \quad (3.15)$$

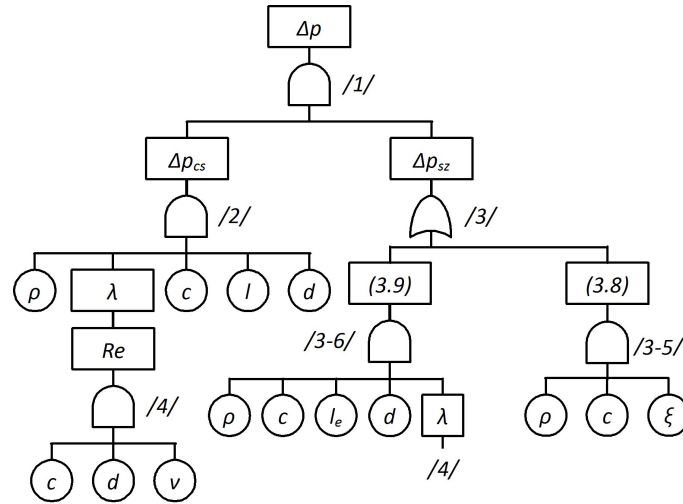


Figure 9. Logical Tree of the Case Study.

Knowing the subsets and the structure of the logical tree the possible sets of the needed parameters can be determined. The tree includes one two-input **OR** gate, therefore two sets of needed parameters can be identified:

$$x_A = x_2 \cup x_4 \cup x_{3-5} = \{\rho; c; l; d; v; \xi\} \quad (3.16)$$

$$x_A = x_2 \cup x_4 \cup x_{3-6} = \{\rho; c; l; d; v; l_e\} \quad (3.17)$$

It means that we should know either parameters from set x_A or from set x_B to set up and apply the mathematical model of the illustrative engineering problem. Fundamentally, the set x_A set was applied in the publication (Pokorádi & Molnár, 2011).

4. Conclusion

A logical tree method has been developed for the determination of possible sets of needed parameters for setting up of a mathematical model or solving an engineering calculation task. The method that named LogTreeMM is theoretically analogous with the Fault Tree Analysis used in system reliability assessment and quality management. The determined logical trees or their parts can be used as blocks to describe the required parameters in complex engineering calculation. In the education the LogTreeMM method can be used for developing of logical engineering thinking of students or pupils.

The Authors prospective scientific research related to this field of applied mathematics and engineering education includes the study of methodologies regarding technical system modeling and its decision making application in field of technical management.

References

- Markowski, Adam S. and Agata Kotynia (2011). "Bow-tie" model in layer of protection analysis. *Process Safety and Environmental Protection* **89**(4), 205 – 213.
- Pokorádi, L. (2011). Sensitivity investigation of fault tree analysis with matrix-algebraic method. *Theory and Applications of Mathematics and Computer Science* **1**(1), 34–44.
- Pokorádi, L. and B. Molnár (2011). Monte-Carlo simulation of the pipeline system to investigate water temperature's effects. *Polytechnical University of Bucharest. Scientific Bulletin. Series D: Mechanical Engineering* **73**(4), 223–236.
- Stamatelatos, M. and J. Caraballo (2002). *Fault Tree Handbook with Aerospace Applications, Office of safety and mission assurance NASA headquarters*. NASA: Washington DC.
- Tchorzewska-Cieslak, B. and K. Boryczko (2010). Relaxed LMI conditions for closed-loop fuzzy systems with tensor-product structure. *Engineering Applications of Artificial Intelligence* pp. 309–320.
- Vesely, W.E., Goldberg F.F., Norman R. and Haasl D. (1987). *Fault tree handbook*, Government Printing Office. Government Printing Office: Washington DC.



Luhn Prime Numbers

Octavian Cira^{a,*}, Florentin Smarandache^b

^aDepartment of Mathematics and Computer Science, "Aurel Vlaicu" University of Arad, România.

^bMathematics & Science Department, University of New Mexico, USA.

Abstract

The first prime number with the special property that its addition with reversal gives as result a prime number too is 229. The prime numbers with this property will be called *Luhn prime numbers*. In this article we intend to present a performing algorithm for determining the *Luhn prime numbers*. Using the presented algorithm all the 50598 *Luhn prime numbers* have been, for p prime smaller than $2 \cdot 10^7$.

Keywords: Prime numbers, Reversal number, Smarandache's function, Luhn prime numbers.

2010 MSC: 11B83.

1. Introduction

The number 229 is the smallest prime number that added with his reverse gives as result a prime number, too. As $1151 = 229 + 922$ is prime.

The first that noted this special property the number 229 has, was Norman Luhn (after 9 February 1999), on the *Prime Curios* website (Caldwell & Honacher Jr., 2014). The prime numbers with this property will be later called *Luhn prime numbers*.

In the *Whats Special About This Number?* list (Friedman, 2014), a list that contains all the numbers between 1 and 9999; beside the number 229 is mentioned that his most important property is that, adding with reversal the resulting number is prime too.

The *On-Line Encyclopedia of Integer Sequences*, (Sloane, 2014, A061783), presents a list 1000 *Luhn prime numbers*. We owe this list to Harry J. Smith, since 28 July 2009. On the same website it is mentioned that Harvey P. Dale published on 27 November 2010 a list that contains 3000 *Luhn prime numbers* and Bruno Berselli published on 5 August 2013 a list that contains 2400 *Luhn prime numbers*.

*Corresponding author

Email addresses: octavian.cira@uav.ro (Octavian Cira), fsmarandache@gmail.com (Florentin Smarandache)

2. Smarandache's function

The function $\mu : \mathbb{N}^* \rightarrow \mathbb{N}^*$, $\mu(n) = m$, where m is the smallest natural number with the property that n divides $m!$ (or $m!$ is a multiple of n) is known in the specialty literature as Smarandache's function, (Smarandache, 1980, 1999; Sondow & Weisstein, 2014). The values resulting from $n = 1, 2, \dots, 18$ are: 1, 2, 3, 4, 5, 3, 7, 4, 6, 5, 11, 4, 13, 7, 5, 6, 17, 6. These values were obtained with an algorithm that results from μ 's definition. The program using this algorithm cannot be used for $n \geq 19$ because the numbers $19!, 20!, \dots$ are numbers which exceed the 17 decimal digits limit and the classic computing model (without the arbitrary precisions arithmetic (Uznanski, 2014)) will generate errors due to the way numbers are represented in the computers memory.

3. Kempner's algorithm

Kempner created an algorithm to calculate $\mu(n)$ using classical factorization $n = p_1^{p_1} \cdot p_2^{p_2} \cdots p_s^{p_s}$, prime number and the generalized numeration base $(\alpha_i)_{[p_i]}$, for $i = \overline{1, s}$, (Kempner, 1918). Partial solutions to the algorithm for $\mu(n)$'s calculation have been given earlier by Lucas and Neuberg, (Sondow & Weisstein, 2014).

Remark. If $n \in \mathbb{N}^*$, n can be decomposed in a product of prime numbers $n = p_1^{\alpha_1} \cdot p_2^{\alpha_2} \cdots p_s^{\alpha_s}$, where p_i are prime numbers so that $p_1 < p_2 < \dots < p_s$, and $s \geq 1$, thus Kempner's algorithm for calculating the μ function is.

$$\mu(n) = \max \left\{ p_1 \cdot (\alpha_{1[p_1]})_{(p_1)}, p_2 \cdot (\alpha_{2[p_2]})_{(p_2)}, \dots, p_s \cdot (\alpha_{s[p_s]})_{(p_s)} \right\},$$

where by $(\alpha_{[p]})_{(p)}$ we understand that α is "written" in the numeration base p (noted $\alpha_{[p]}$) and it is "read" in the p numeration base (noted $\beta_{(p)}$, where $\beta = \alpha_{[p]}$), (Smarandache, 1999, p. 39).

4. Programs

The list of prime numbers was generated by a program that uses the Sieve of Eratosthenes the linear version of Pritchard, Pritchard (1987), which is the fastest algorithm to generate prime numbers until the limit of L , if $L \leq 10^8$. The list of prime numbers until to $2 \cdot 10^7$ is generated in about 5 seconds. For the limit $L > 10^8$ the fastest algorithm for generating the prime numbers is the Sieve of Atkin, Atkin & Bernstein (2004).

Program 4.1. The Program for the Sieve of Eratosthenes, the linear version of Pritchard using minimal memory space is:

```
CEPbm(L) :=  $\lambda \leftarrow \text{floor}\left(\frac{L}{2}\right)$ 
              for  $k \in 1.. \lambda$ 
                 $is\_prime_k \leftarrow 1$ 
               $prime \leftarrow (2\ 3\ 5\ 7)^T$ 
               $i \leftarrow \text{last}(prime) + 1$ 
              for  $j \in 4, 7.. \lambda$ 
                 $is\_prime_j \leftarrow 0$ 
```

```

k ← 3
s ← (primek-1)2
t ← (primek)2
while t ≤ L
    for j ∈ t, t + 2 · primek..L
        is_prime $\frac{j-1}{2}$  ← 0
    for j ∈ s + 2, s + 4..t - 2
        if is_prime $\frac{j-1}{2}$  = 1
            primei ← j
            i ← i + 1
    s ← t
    k ← k + 1
    t ← (primek)2
for j ∈ s + 2, s + 4..L
    if is_prime $\frac{j-1}{2}$  = 1
        primei ← j
        i ← i + 1
return prime

```

Program 4.2. The factorization program of a natural number; this program uses the vector p representing prime numbers, generated with the Sieve of Eratosthenes. The Sieve of Eratosthenes is called upon through the following sequence:

$$L := 2 \cdot 10^7 \quad t_0 = \text{time}(0) \quad p := \text{CEPbm}(L) \quad t_1 = \text{time}(1)$$

$$(t_1 - t_0)s = 5.064s \quad \text{last}(p) = 1270607 \quad p_{\text{last}(p)} = 19999999$$

```

Fa(m) := return ("m = " m " > ca ultimul p2") if m > (plast(p))2
j ← 1
k ← 0
f ← (1 1)
while m ≥ pj
    if mod(m, pj) = 0
        k ← k + 1
        m ←  $\frac{m}{p_j}$ 
    otherwise
        f ← stack[f, (pj, k)] if k > 0
        j ← j + 1
        k ← 0
f ← stack[f, (pj, k)] if k > 0
return submatrix(f, 2, rows(f), 1, 2)

```

We presented the Kempner's algorithm using Mathcad programs required for the algorithm.

Program 4.3. The function counting all the digits in the p base of numeration in which is n .

$$ncb(n, p) := \begin{cases} \text{return } \text{ceil}(\log(n, p)) & \text{if } n > 1 \\ \text{return } 1 & \text{otherwise} \end{cases}$$

Where $\text{ceil}(x)$ is a Mathcad function which gives the smallest integer $\geq x$ and $\log(n, p)$ is logarithm in base p from n .

Program 4.4. The program intended to generate the p generalized base of numeration (noted by Smarandache $[p]$) for a number with m digits.

$$a(p, m) := \begin{cases} \text{for } i \in 1..m \\ \quad a_i \leftarrow \frac{p^i - 1}{p - 1} \\ \text{return } a \end{cases}$$

Program 4.5. The program intended to generate for the p base of numeration (noted by Smarandache (p)) to write the α number.

$$b(\alpha, p) := \begin{cases} \text{return } (1) & \text{if } p = 1 \\ \text{for } i \in 1..ncb(\alpha, p) \\ \quad b_i \leftarrow p^{i-1} \\ \text{return } b \end{cases}$$

Program 4.6. Program that determines the digits of the generalized base of numeration $[p]$ for the number n .

$$Nbg(n, p) := \begin{cases} m \leftarrow ncb(n, p) \\ a \leftarrow a(p, m) \\ \text{return } (1) & \text{if } m=0 \\ \text{for } i \in m..1 \\ \quad \begin{cases} c_i \leftarrow \text{trunc}\left(\frac{n}{a_i}\right) \\ n \leftarrow \text{mod}(n, a_i) \end{cases} \\ \text{return } c \end{cases}$$

Where $\text{trunc}(x)$ returns the integer part of x by removing the fractional part, and $\text{mod}(x, y)$ returns the remainder on dividing x by y (x modulo y).

Program 4.7. Program for Smarandache's function.

$$\mu(n) := \begin{cases} \text{return "Err. } n \text{ nu este intreg"} & \text{if } n \neq \text{trunc}(n) \\ \text{return "Err. } n < 1" & \text{if } n < 1 \\ \text{return } (1) & \text{if } n=1 \\ f \leftarrow Fa(n) \\ p \leftarrow f^{(1)} \\ \alpha \leftarrow f^{(2)} \\ \text{for } k = 1..rows(p) \\ \quad \eta_k \leftarrow p_k \cdot Nbg(\alpha_k, p_k) \cdot b(\alpha_k, p_k) \\ \text{return } \max(\eta) \end{cases}$$

This program calls the $Fa(n)$ factorization with prime numbers. The program uses the Smarandache's Remark 3 – about the Kempner algorithm. The $\mu.prn$ file generation is done once. The reading of this generated file in Mathcad's documents results in a great time–save.

Program 4.8. Program with which the file $\mu.prn$ is generated

```
VFμ(N) := | μ1 ← 1
           | for n ∈ 2..N
           |   μn ← μ(n)
           | return μ
```

This program calls the program 4.7 for calculating the value of the μ function. The sequence of the $\mu.prn$ file generation is:

$$t_0 := \text{time}(0) \quad \text{WRITEPRN}(\text{"}\mu.prn\text{"}) := \text{VF}\mu(2 \cdot 10^7) \quad t_1 := \text{time}(1)$$

$$(t_1 - t_0)\text{sec} = \text{"}5 : 17 : 32.625\text{"}hhmmss$$

Smarandache's function is important because it characterizes prime numbers – through the following fundamental property:

Theorem 4.9. Let be p an integer > 4 , than p is prime number if and only if $\mu(p) = p$.

Proof. See (Smarandache, 1999, p. 31). □

Hence, the fixed points of this function are prime numbers (to which is added 4). Due to this property the function is used as primality test.

Program 4.10. Program for testing μ 's primality. This program returns the 0 value if the number is not prime number and the 1 value if the number is a prime. The file $\mu.prn$ will be read and it will be assigned to the μ vector.

$$\text{ORIGIN} := 1 \quad \mu := \text{READPRN}(\text{"}\dots \backslash \mu.prn\text{"})$$

```
Tpμ(n) := | return "Err. n < 1 sau n ∉ ℤ" if n < 1 ∨ n ≠ trunc(n)
           | if n > 4
           |   | return 0 if μn ≠ n
           |   | return 1 otherwise
           | otherwise
           |   | return 0 if n=1 ∨ n=4
           |   | return 1 otherwise
```

Program 4.11. Program that provides the reverses of the given m number.

$$\begin{aligned}
 R(m) := & \left| \begin{array}{l} n \leftarrow \text{floor}(\log(m)) \\ x \leftarrow m \cdot 10^{-n} \\ \text{for } k \in 1..n \\ \quad \left| \begin{array}{l} c_k \leftarrow \text{trunc}(x) \\ x \leftarrow (x - c_k) \cdot 10 \end{array} \right. \\ c_{n+1} \leftarrow \text{round}(x) \\ Rm \leftarrow 0 \\ \text{for } k \in n+1..2 \\ \quad Rm \leftarrow (Rm + c_k) \cdot 10 \\ \text{return } Rm + c_1 \end{array} \right.
 \end{aligned}$$

Where $\text{floor}(x)$ returns the greatest integer $\leq x$ and $\text{round}(x)$ returns x rounded to the nearest integer.

Program 4.12. Search program for the Luhn prime numbers.

$$\begin{aligned}
 PLuhn(L) := & \left| \begin{array}{l} n \leftarrow \text{last}(p) \\ S \leftarrow (229) \\ k \leftarrow 51 \\ \text{while } p_k \leq L \\ \quad \left| \begin{array}{l} N \leftarrow R(p_k) + p_k \\ S \leftarrow \text{stack}(S, p_k) \text{ if } T\mu(N) = 1 \\ k \leftarrow k + 1 \end{array} \right. \\ \text{return } S \end{array} \right.
 \end{aligned}$$

The function $\text{stack}(A, B, \dots)$ is applied for merging matrixes top-down. The number of columns in matrixes should also be the same. The discussed functions could be applied to vectors as well.

Execution of the program $PLuhn$ was made with sequence

$$S := PLuhn(2 \cdot 10^7)$$

The initialization of the S stack is done with the vector that contains the number 229. The variable k has the initial value of 51 because the index of the 229 prime number is 50, so that the search for the *Luhn prime numbers* will begin with $p_{51} = 233$.

5. List of prime numbers Luhn

We present a partial list of the 50598 *Luhn prime numbers* smaller than $2 \cdot 10^7$ (the first 321 and the last 120):

229, 239, 241, 257, 269, 271, 277, 281, 439, 443, 463, 467, 479, 499, 613, 641, 653, 661, 673, 677, 683, 691, 811, 823, 839, 863, 881, 20011, 20029, 20047, 20051, 20101, 20161, 20201, 20249, 20269, 20347, 20389, 20399, 20441, 20477, 20479, 20507, 20521, 20611, 20627, 20717, 20759, 20809, 20879, 20887, 20897, 20981, 21001, 21019, 21089, 21157, 21169, 21211, 21377, 21379, 21419, 21467, 21491, 21521, 21529, 21559, 21569, 21577, 21601, 21611, 21617, 21647, 21661,

21701, 21727, 21751, 21767, 21817, 21841, 21851, 21859, 21881, 21961, 21991, 22027, 22031, 22039, 22079, 22091, 22147, 22159, 22171, 22229, 22247, 22291, 22367, 22369, 22397, 22409, 22469, 22481, 22501, 22511, 22549, 22567, 22571, 22637, 22651, 22669, 22699, 22717, 22739, 22741, 22807, 22859, 22871, 22877, 22961, 23017, 23021, 23029, 23081, 23087, 23099, 23131, 23189, 23197, 23279, 23357, 23369, 23417, 23447, 23459, 23497, 23509, 23539, 23549, 23557, 23561, 23627, 23689, 23747, 23761, 23831, 23857, 23879, 23899, 23971, 24007, 24019, 24071, 24077, 24091, 24121, 24151, 24179, 24181, 24229, 24359, 24379, 24407, 24419, 24439, 24481, 24499, 24517, 24547, 24551, 24631, 24799, 24821, 24847, 24851, 24889, 24979, 24989, 25031, 25057, 25097, 25111, 25117, 25121, 25169, 25171, 25189, 25219, 25261, 25339, 25349, 25367, 25409, 25439, 25469, 25471, 25537, 25541, 25621, 25639, 25741, 25799, 25801, 25819, 25841, 25847, 25931, 25939, 25951, 25969, 26021, 26107, 26111, 26119, 26161, 26189, 26209, 26249, 26251, 26339, 26357, 26417, 26459, 26479, 26489, 26591, 26627, 26681, 26701, 26717, 26731, 26801, 26849, 26921, 26959, 26981, 27011, 27059, 27061, 27077, 27109, 27179, 27239, 27241, 27271, 27277, 27281, 27329, 27407, 27409, 27431, 27449, 27457, 27479, 27481, 27509, 27581, 27617, 27691, 27779, 27791, 27809, 27817, 27827, 27901, 27919, 28001, 28019, 28027, 28031, 28051, 28111, 28229, 28307, 28309, 28319, 28409, 28439, 28447, 28571, 28597, 28607, 28661, 28697, 28711, 28751, 28759, 28807, 28817, 28879, 28901, 28909, 28921, 28949, 28961, 28979, 29009, 29017, 29021, 29027, 29101, 29129, 29131, 29137, 29167, 29191, 29221, 29251, 29327, 29389, 29411, 29429, 29437, 29501, 29587, 29629, 29671, 29741, 29759, 29819, 29867, 29989, ...
8990143, 8990209, 8990353, 8990441, 8990563, 8990791, 8990843, 8990881, 8990929, 8990981, 8991163, 8991223, 8991371, 8991379, 8991431, 8991529, 8991553, 8991613, 8991743, 8991989, 8992069, 8992091, 8992121, 8992153, 8992189, 8992199, 8992229, 8992259, 8992283, 8992483, 8992493, 8992549, 8992561, 8992631, 8992861, 8992993, 8993071, 8993249, 8993363, 8993401, 8993419, 8993443, 8993489, 8993563, 8993723, 8993749, 8993773, 8993861, 8993921, 8993951, 8994091, 8994109, 8994121, 8994169, 8994299, 8994463, 8994473, 8994563, 8994613, 8994721, 8994731, 8994859, 8994871, 8994943, 8995003, 8995069, 8995111, 8995451, 8995513, 8995751, 8995841, 8995939, 8996041, 8996131, 8996401, 8996521, 8996543, 8996651, 8996681, 8996759, 8996831, 8996833, 8996843, 8996863, 8996903, 8997059, 8997083, 8997101, 8997463, 8997529, 8997553, 8997671, 8997701, 8997871, 8997889, 8997931, 8997943, 8997979, 8998159, 8998261, 8998333, 8998373, 8998411, 8998643, 8998709, 8998813, 8998919, 8999099, 8999161, 8999183, 8999219, 8999311, 8999323, 8999339, 8999383, 8999651, 8999671, 8999761, 8999899, 8999981.

6. Conclusions

The list of all *Luhn prime numbers*, that totaled 50598 numbers, was determined within a time span of 54 seconds, on an Intel processor of 2.20 GHz.

References

- Atkin, A. O. L. and D. J. Bernstein (2004). Prime Sieves Using Binary Quadratic Forms. *Math. Comp.* **73**, 1023–1030.
- Caldwell, Ch. K. and G. L. Honacher Jr. (2014). Prime Curios! The Dictionary of Prime Number Trivia.
- Friedman, E. (2014). What's Special About This Number? From: Erich's Place.

Kempner, A. J. (1918). Miscellanea. *Amer. Math. Monthly* **25**, 201–210.

Pritchard, P. (1987). Linear prime number sieves: a family tree. *Sci. Comp. Prog.* **9**(1), 17–35.

Sloane, N. J. A (2014). Primes p such that $p + (p \text{ reversed})$ is also a prime. From: The On-Line Encyclopedia of Integer Sequences.

Smarandache, F. (1980). O nouă funcție în teoria analitică a numerelor. *An. Univ. Timișoara* **XVIII**(fasc. 1), 79–88.

Smarandache, F. (1999). *Asupra unor noi funcții în teoria numerelor*. Universitatea de Stat Moldova. Chișinău, Republica Moldova.

Sondow, J. and E. W. Weisstein (2014). Smarandache Function.

Uznanski, D. (2014). Arbitrary precision.



The Applicability of $\$$ -Calculus to Solve Some Turing Machine Undecidable Problems

Eugene Eberbach^{a,*}

^a*Department of Engineering and Science, Rensselaer Polytechnic Institute,
Hartford, 275 Windsor Street, CT 06120, United States.*

Abstract

The $\$$ -calculus process algebra for problem solving applies the cost performance measures to converge in finite time or in the limit to optimal solutions with minimal problem solving costs. The $\$$ -calculus belongs to superTuring models of computation. Its main goal is to provide the support to solve hard computational problems. It allows also to solve in the limit some undecidable problems. In the paper we demonstrate how to solve in the limit Turing Machine Halting Problem, to approximate the universal search algorithm, to decide diagonalization language, nontrivial properties of recursively enumerable languages, and how to solve Post Correspondence Problem and Busy Beaver Problem.

Keywords: problem solving, hypercomputation, expressiveness, superTuring models of computation, resource bounded computation, process algebras, $\$$ -calculus.

2010 MSC: primary classification: 68Qxx, 68Txx.

2012 CCS: primary classification: Theory of computation, mathematics of computing. Secondary classifications: models of computation, Turing machines, concurrency, process calculi, formal languages and automata theory, formalisms, automata over infinite objects, mathematical optimization, discrete optimization.

1. Introduction

In this paper, the expressiveness of the $\$$ -calculus process algebra of bounded rational agents (Eberbach, 1997, 2005a, 2006, 2007) is investigated. The $\$$ -calculus, presented in this paper, belongs to superTuring models of computation and provides a support to handle intractability and undecidability in problem solving. In the paper, we present the applicability of $\$$ -calculus to solve (in hypercomputational sense) some undecidable problems.

The paper is organized as follows. In section 2, we briefly recall some basic notions related to Turing machine problem solving and hypercomputation. In section 3, we outline the $\$$ -calculus

*Corresponding author

Email address: eberbe@rpi.edu (Eugene Eberbach)

process algebra of bounded rational agents. In section 4, we present the solution of the halting problem of Universal Turing Machine, and approximate solution of the universal search algorithm. In section 5, other TM unsolvable problems are investigated, including the diagonalization language, nontrivial properties of recursively enumerable languages, Post Correspondence Problem and Busy Beaver Problem. Section 6 contains conclusions.

2. Problem Solving in Turing Machines and Hypercomputation

Turing Machines (TMs) (Turing, 1937, 1939) and algorithms are two fundamental concepts of computer science and problem solving. Turing Machines describe the limits of problem solving using conventional recursive algorithms, and laid the foundation of current computer science in the 1960s.

Note that there are several other models of algorithms, called super-recursive algorithms, that can compute more than Turing Machines, using hypercomputational/superTuring models of computation (Burgin, 2004; Syropoulos, 2007). The battle between reductionists (believing in strong Church-Turing Thesis and “unsinkability” of Turing machine model) and remodelers (hyper-computationalists trying to develop new super-Turing models of computation for solution of Turing Machine undecidable problems) is not over, however shifting gradually in favor of hyper-computationalists (Aho, 2011; Cooper, 2012; Wegner *et al.*, 2012).

It turns out that (TM) *undecidable problems* cannot be solved by TMs and *intractable problems* are solvable, but require too many resources (e.g., steps or memory). For undecidable problems effective recipes do not exist - problems are called nonalgorithmic or nonrecursive. On the other hand, for intractable problems algorithms exist, but running them on a deterministic Turing Machine, requires an exponential amount of time (the number of elementary moves of the TM) as a function of the TM input.

We use the simplicity of the TM model to prove formally that there are specific problems (languages) that the TM cannot solve (Hopcroft *et al.*, 2001). Solving the problem is equivalent to decide whether a string belongs to the language. A problem that cannot be solved by computer (Turing machine) is called *undecidable* (TM-undecidable). The class of languages accepted by Turing machines are called *recursively enumerable (RE-) languages*. For RE-languages, TM can accept the strings in the language but cannot tell for certain that a string is not in the language.

There are two classes of Turing machine unsolvable languages (problems):

recursively enumerable RE but not recursive - TM can accept the strings in the language but cannot tell for certain that a string is not in the language (e.g., the language of the universal Turing machine, or Post’s Correspondence Problem language). A language is decidable but its complement is undecidable, or vice versa: a language is undecidable but its complement is decidable.

non-RE - no TM can even recognize the members of the language in the RE sense (e.g., the diagonalization language). Neither a language nor its complement is decidable.

Decidable problems have a (recursive) algorithm, i.e., TM halts whether or not it accepts its input. Decidable problems are described by *recursive languages*. Algorithms as we know are associated

with the class of recursive languages, a subset of recursively enumerable languages for which we can construct its accepting TM. For recursive languages, both a language and its complement are decidable.

Turing Machines are used as a formal model of classical (recursive) algorithms. An algorithm should consist of a finite number of steps, each having well defined and implementable meaning. We are convinced that computer computations are not restricted to such restrictive definition of algorithms only.

Definition 2.1. By superTuring computation (*also called hypercomputation*) we mean any computation that cannot be carried out by a Turing Machine as well as any (algorithmic) computation carried out by a Turing Machine.

In (Eberbach & Wegner, 2003; Eberbach *et al.*, 2004), several superTuring models have been discussed and overviewed. The incomplete list includes Turing’s o-machines, c-machines and u-machines, cellular automata, discrete and analog neural networks, Interaction Machines, Persistent Turing Machines, Site and Internet Machines, the π -calculus, the $\$$ -calculus, Inductive Turing Machines, Infinite Time Turing Machines, Accelerating Turing Machines and Evolutionary Turing Machines. In particular, the author proposed two superTuring models of computation: the $\$$ -Calculus (Eberbach, 2005a, 2007) and Evolutionary Turing Machine (Eberbach, 2005b; Eberbach & Burgin, 2009).

SuperTuring models derive their higher than the TM expressiveness using three principles: *interaction*, *evolution*, or *infinity*. In the *interaction principle* the model becomes open and the agent interacts with either a more expressive component or with an infinite many components. In the *evolution principle*, the model can evolve to a more expressive one using non-recursive variation operators. In the *infinity principle*, models can use unbounded resources: time, memory, the number of computational elements, an unbounded initial configuration, an infinite alphabet, etc. The details can be found in (Eberbach & Wegner, 2003; Eberbach *et al.*, 2004).

3. The $\$$ -Calculus Algebra of Bounded Rational Agents

The $\$$ -calculus is a mathematical model of processes capturing both the final outcome of problem solving as well as the interactive incremental way how the problems are solved. The $\$$ -calculus is a process algebra of Bounded Rational Agents for interactive problem solving targeting intractable and undecidable problems. It has been introduced in the late of 1990s (Eberbach, 1997, 2005a, 2007). The $\$$ -calculus (pronounced cost calculus) is a formalization of resource-bounded computation (also called anytime algorithms), proposed by Dean, Horvitz, Zilberstein and Russell in the late 1980s and early 1990s (Horvitz & Zilberstein, 2001; Russell & Norvig, 2002). Anytime algorithms are guaranteed to produce better results if more resources (e.g., time, memory) become available. The standard representative of process algebras, the π -calculus (Milner *et al.*, 1992; Milner, 1999) is believed to be the most mature approach for concurrent systems.

The $\$$ -calculus rests upon the primitive notion of *cost* in a similar way as the π -calculus was built around a central concept of *interaction*. Cost and interaction concepts are interrelated in the sense that cost captures the quality of an agent interaction with its environment. The unique feature of the $\$$ -calculus is that it provides a support for problem solving by incrementally searching for

solutions and using cost to direct its search. The basic $\$$ -calculus search method used for problem solving is called $k\Omega$ -optimization. The $k\Omega$ -optimization represents this “impossible” to construct, but “possible to approximate indefinitely” universal algorithm. It is a very general search method, allowing the simulation of many other search algorithms, including A*, minimax, dynamic programming, tabu search, or evolutionary algorithms. Each agent has its own Ω search space and its own limited horizon of deliberation with depth k and width b . Agents can cooperate by selecting actions with minimal costs, can compete if some of them minimize and some maximize costs, and be impartial (irrational or probabilistic) if they do not attempt optimize (evolve, learn) from the point of view of the observer. It can be understood as another step in the never ending dream of universal problem solving methods recurring throughout all computer science history. The $\$$ -calculus is applicable to robotics, software agents, neural nets, and evolutionary computation. Potentially it could be used for design of cost languages, cellular evolvable cost-driven hardware, DNA-based computing and molecular biology, electronic commerce, and quantum computing. The $\$$ -calculus leads to a new programming paradigm *cost languages* and a new class of computer architectures *cost-driven computers*.

3.1. The $\$$ -Calculus Syntax

In $\$$ -calculus everything is a cost expression: agents, environment, communication, interaction links, inference engines, modified structures, data, code, and meta-code. $\$$ -expressions can be simple or composite. Simple $\$$ -expressions α are considered to be executed in one atomic indivisible step. Composite $\$$ -expressions P consist of distinguished components (simple or composite ones) and can be interrupted.

Definition 3.1. The $\$$ -calculus The set \mathcal{P} of $\$$ -calculus process expressions consists of simple $\$$ -expressions α and composite $\$$ -expressions P , and is defined by the following syntax:

α	$::=$	$(\$_{i \in I} P_i)$	cost
		$(\rightarrow_{i \in I} c P_i)$	send P_i with evaluation through channel c
		$(\leftarrow_{i \in I} c X_i)$	receive X_i from channel c
		$(\prime_{i \in I} P_i)$	suppress evaluation of P_i
		$(a_{i \in I} P_i)$	defined call of simple $\$$ -expression a with parameters P_i , and and its optional associated definition $(:= (a_{i \in I} X_i) < R >)$ with body R evaluated atomically
		$(\bar{a}_{i \in I} P_i)$	negation of defined call of simple $\$$ -expression a
P	$::=$	$(\circ_{i \in I} \alpha P_i)$	sequential composition
		$(\parallel_{i \in I} P_i)$	parallel composition
		$(\cup_{i \in I} P_i)$	cost choice
		$(\sqcup_{i \in I} P_i)$	adversary choice
		$(\sqcup_{i \in I} P_i)$	general choice
		$(f_{i \in I} P_i)$	defined process call f with parameters P_i , and its associated definition $(:= (f_{i \in I} X_i) R)$ with body R (normally suppressed); $(^1 R)$ will force evaluation of R exactly once

The indexing set I is a possibly countably infinite. In the case when I is empty, we write empty parallel composition, general, cost and adversary choices as \perp (blocking), and empty sequential composition (I empty and $\alpha = \varepsilon$) as ε (invisible transparent action, which is used to mask, make invisible parts of $\$$ -expressions). Adaptation (evolution/upgrade) is an essential part of $\$$ -calculus, and all $\$$ -calculus operators are infinite (an indexing set I is unbounded). The $\$$ -calculus agents interact through send-receive pair as the essential primitives of the model.

Sequential composition is used when $\$$ -expressions are evaluated in a textual order. Parallel composition is used when expressions run in parallel and it picks a subset of non-blocked elements at random. Cost choice is used to select the cheapest alternative according to a cost metric. Adversary choice is used to select the most expensive alternative according to a cost metric. General choice picks one non-blocked element at random. General choice is different from cost and adversary choices. It uses guards satisfiability. Cost and adversary choices are based on cost functions. Call and definition encapsulate expressions in a more complex form (like procedure or function definitions in programming languages). In particular, they specify recursive or iterative repetition of $\$$ -expressions.

Simple cost expressions execute in one atomic step. Cost functions are used for optimization and adaptation. The user is free to define his/her own cost metrics. Send and receive perform handshaking message-passing communication, and inferencing. The suppression operator suppresses evaluation of the underlying $\$$ -expressions. Additionally, a user is free to define her/his own simple $\$$ -expressions, which may or may not be negated.

3.2. The $\$$ -Calculus Semantics: The $k\Omega$ -Search

In this section we define the operational semantics of the $\$$ -calculus using the $k\Omega$ -search that captures the dynamic nature and incomplete knowledge associated with the construction of the problem solving tree.

The basic $\$$ -calculus problem solving method, the $k\Omega$ -optimization, is a very general search method providing meta-control, and allowing to simulate many other search algorithms, including A^* , minimax, dynamic programming, tabu search, or evolutionary algorithms (Russell & Norvig, 2002). The problem solving works iteratively: through select, examine and execute phases. In the select phase the tree of possible solutions is generated up to k steps ahead, and agent identifies its alphabet of interest for optimization Ω . This means that the tree of solutions may be incomplete in width and depth (to deal with complexity). However, incomplete (missing) parts of the tree are modeled by silent $\$$ -expressions ε , and their cost estimated (i.e., not all information is lost). The above means that $k\Omega$ -optimization may be if some conditions are satisfied to be complete and optimal. In the examine phase the trees of possible solutions are pruned minimizing cost of solutions, and in the execute phase up to n instructions are executed. Moreover, because the $\$$ operator may capture not only the cost of solutions, but the cost of resources used to find a solution, we obtain a powerful tool to avoid methods that are too costly, i.e., the $\$$ -calculus directly minimizes search cost. This basic feature, inherited from anytime algorithms, is needed to tackle directly hard optimization problems, and allows to solve total optimization problems (the best quality solutions with minimal search costs). The variable k refers to the limited horizon for optimization, necessary due to the unpredictable dynamic nature of the environment. The variable Ω refers to a reduced alphabet of information. No agent ever has reliable information about all factors that

influence all agents behavior. To compensate for this, we mask factors where information is not available from consideration; reducing the alphabet of variables used by the $\$$ -function. By using the $k\Omega$ -optimization to find the strategy with the lowest $\$$ -function, meta-system finds a satisficing solution, and sometimes the optimal one. This avoids wasting time trying to optimize behavior beyond the foreseeable future. It also limits consideration to those issues where relevant information is available. Thus the $k\Omega$ optimization provides a flexible approach to local and/or global optimization in time or space. Technically this is done by replacing parts of $\$$ -expressions with invisible $\$$ -expressions ε , which remove part of the world from consideration (however, they are not ignored entirely - the cost of invisible actions is estimated).

The $k\Omega$ -optimization meta-search procedure can be used both for single and multiple cooperative or competitive agents working online ($n \neq 0$) or offline ($n = 0$). The $\$$ -calculus programs consist of multiple $\$$ -expressions for several agents.

Let's define several auxiliary notions used in the $k\Omega$ -optimization meta-search. Let:

- \mathcal{A} - be an alphabet of $\$$ -expression names for an enumerable *universe of agent population* (including an environment, i.e., one agent may represent an environment). Let $\mathcal{A} = \bigcup_i A_i$, where A_i is the alphabet of $\$$ -expression names (simple or complex) used by the i -th agent, $i = 1, 2, \dots, \infty$. We will assume that the names of $\$$ -expressions are unique, i.e., $A_i \cap A_j = \emptyset, i \neq j$ (this always can be satisfied by indexing $\$$ -expression name by a unique agent index. This is needed for an agent to execute only own actions). The agent population size will be denoted by $p = 1, 2, \dots, \infty$.
- $x_i[0] \in \mathcal{P}$ - be an initial $\$$ -expression for the i -th agent, and its initial search procedure $k\Omega_i[0]$.
- $\min(\$_i(k\Omega_i[t], x_i[t]))$ - be an implicit default goal and $Q_i \subseteq \mathcal{P}$ be an optional (explicit) goal. The default goal is to find a pair of $\$$ -expressions, i.e., any pair $(k\Omega_i[t], x_i[t])$ being

$$\min\{(\$_i(k\Omega_i[t], x_i[t])) = \$_{1i}(\$_{2i}(k\Omega_i[t]), \$_{3i}(x_i[t]))\},$$

where $\$_{3i}$ is a problem-specific cost function, $\$_{2i}$ is a search algorithm cost function, and $\$_{1i}$ is an aggregating function combining $\$_{2i}$ and $\$_{3i}$. This is the default goal for total optimization looking for the best solutions $x_i[t]$ with minimal search costs $k\Omega_i[t]$. It is also possible to look for the optimal solution only, i.e., the best $x_i[t]$ with minimal value of $\$_{i3}$, or the best search algorithm $k\Omega_i[t]$ with minimal costs of $\$_{i2}$. The default goal can be overwritten or supplemented by any other termination condition (in the form of an arbitrary $\$$ -expression Q) like the maximum number of iterations, the lack of progress, etc.

- $\$$ - a cost function performance measure (selected from the library or user defined). It consists of the problem specific cost function $\$_{3i}$, a search algorithm cost function $\$_{2i}$, and an aggregating function $\$_{1i}$. Typically, a user provides cost of simple $\$$ -expressions or an agent can learn such costs (e.g., by reinforcement learning). The user selects or defines also how the costs of composite $\$$ -expressions will be computed. The cost of the solution tree is the function of its components: costs of nodes (states) and edges (actions). This allows to express both the quality of solutions and search cost.

- $\Omega_i \subseteq \mathcal{A}$ - a *scope of deliberation/interests of the i -th agent*, i.e., a subset of the universe's of \mathcal{A} -expressions chosen for optimization. All elements of $\mathcal{A} - \Omega_i$ represent irrelevant or unreachable parts of an environment, of a given agent or other agents, and will become invisible (replaced by ε), thus either ignored or unreachable for a given agent (makes optimization local spatially). Expressions over $\mathcal{A} - \Omega_i$ will be treated as observationally congruent (cost of ε will be neutral in optimization, e.g., typically set to 0). All expressions over $\Omega_i - \mathcal{A}$ will be treated as strongly congruent - they will be replaced by ε and although invisible, their cost will be estimated using the best available knowledge of an agent (may take arbitrary values from the cost function domain).
- $b_i = 0, 1, 2, \dots, \infty$ - a branching factor of the search tree (LTS), i.e., the maximum number of generated children for a parent node. For example, hill climbing has $b_i = 1$, for binary tree $b_i = 2$, and $b_i = \infty$ is a shorthand to mean to generate all children (possibly infinite many).
- $k_i = 0, 1, 2, \dots, \infty$ - represents *the depth of deliberation*, i.e., the number of steps in the derivation tree selected for optimization in the examine phase (decreasing k_i prevents combinatorial explosion, but can make optimization local in time). $k_i = \infty$ is a shorthand to mean to the end to reach a goal (may not require infinite number of steps). $k_i = 0$ means omitting optimization (i.e., the empty deliberation) leading to reactive behaviors. Similarly, a branching factor $b_i = 0$ will lead to an empty deliberation too. Steps consist of multi-sets of simple \mathcal{A} -expressions, i.e., a parallel execution of one or more simple \mathcal{A} -expressions constitutes one elementary step.
- $n_i = 0, 1, 2, \dots, \infty$ - the number of steps selected for execution in the execute phase. For $n_i > k_i$ steps larger than k_i will be executed without optimization in reactive manner. For $n_i = 0$ execution will be postponed until the goal will be reached.
For the depth of deliberation $k_i = 0$, the $k\Omega$ -search will work in the style of imperative programs (reactive agents), executing up to n_i consecutive steps in each loop iteration. For $n_i = 0$ search will be offline, otherwise for $n_i \neq 0$ - online.
- *gp, reinf, strongcon, update* - auxiliary flags used in the $k\Omega$ -optimization meta-search procedure.

Each agent has its own $k\Omega$ -search procedure $k\Omega_i[t]$ used to build the solution $x_i[t]$ that takes into account other agent actions (by selecting its alphabet of interests Ω_i that takes actions of other agents into account). Thus each agent will construct its own view of the whole universe that only sometimes will be the same for all agents (this is an analogy to the subjective view of the “objective” world by individuals having possibly different goals and different perception of the universe).

Definition 3.2. The $k\Omega$ -Optimization Meta-Search Procedure The $k\Omega$ - optimization meta-search procedure $k\Omega_i[t]$ for the i -th agent, $i = 0, 1, 2, \dots$, from an enumerable universe of agent population and working in time generations $t = 0, 1, 2, \dots$ is a complex \mathcal{A} -expression (meta-procedure) consisting of simple \mathcal{A} -expressions $init_i[t]$, $sel_i[t]$, $exam_i[t]$, $goal_i[t]$, $\$i[t]$, complex \mathcal{A} -expression $loop_i[t]$ and $exec_i[t]$, and constructing solutions, its input $x_i[t]$, from predefined and user defined simple

and complex $\$$ -expressions. For simplicity, we will skip time and agent indices in most cases if it does not cause confusion, and we will write *init*, *loop*, *sel*, *exam*, *goal_i* and $\$_i$. Each *i*-th agent performs the following $k\Omega$ -search procedure $k\Omega_i[t]$ in the time generations $t = 0, 1, 2, \dots$:

```
(:= (kΩi[t] xi[t]) (◦ (init (kΩi[0] xi[0])) // initialize kΩi[0] and xi[0]
  (loop xi[t + 1])) // basic cycle: select, examine,
) // execute
```

where *loop* meta- $\$$ -expression takes the form of the select-examine-execute cycle performing the $k\Omega$ -optimization until the goal is satisfied. At that point, the agent re-initializes and works on a new goal in the style of the never ending reactive program:

```
(:= (loop xi[t]) // loop recursive definition
  (⊔ (◦ (goali[t] (kΩi[t] xi[t])) // goal not satisfied, default goal
    // min($i (kΩi[t] xi[t]))
    (sel xi[t]) // select: build problem solution tree k step
    // deep, b wide
    (exam xi[t]) // examine: prune problem solution tree in
    // cost ∪ and in adversary ∪ choices
    (exec (kΩi[t] xi[t])) // execute: run optimal xi n steps and
    // update kΩi parameters
    (loop xi[t + 1])) // return back to loop
  (◦ (goali[t] (kΩi[t] xi[t])) // goal satisfied - re-initialize search
    (kΩi[t] xi[t])))
)
```

Simple $\$$ -expressions *init*, *sel*, *exam*, *goal* with their atomically executed bodies are defined below. On the other hand, *exec* can be interrupted after each action, thus it is not atomic.

1. **Initialization** ($:=$ (*init* ($k\Omega_i[0]$ $x_i[0]$)) $<$ *init_body* $>$): where *init_body* = ($\circ (\leftarrow_{i \in I}$ *user_channel* X_i) (\sqcup *cond_init* (\circ *cond_init* (*init_body*))), and *cond_init* = (\sqcup ($x_i[0] = \perp$) ($k_i = n_i = 0$)), and successive X_i , $i = 1, 2, \dots$ will be the following: $k\Omega_i[0]$ an initial meta-search procedure (default: as provided in this definition), $k_i, b_i, n_i, \Omega_i, A_i$ (defaults: $k_i = b_i = n_i = \infty$, $\Omega_i = A_i = \mathcal{A}$); simple and complex $\$$ -expressions definitions over $A_i \cup \Omega_i$ (default: no definitions);

initialize costs of simple $\$$ -expressions randomly and set reinforcement learning flag *rein_f* = 1 (default: get costs of simple $\$$ -expressions from the user; *rein_f* = 0); $\$_{i1}$ an aggregating cost function (default: addition), $\$_{i2}$ and $\$_{i3}$ search and solution specific cost functions (default: a standard cost function as defined in the next section);

Q_i optional goal of computation (default: $\min(\$_i (k\Omega_i[t], x_i[t]))$);

$x_i[0]$ an initial $\$$ -expression solution (an initial state of LTS for the *i*-th agent) over alphabet $A_i \cup \Omega_i$. This resets *gp_i* flag to 0 (default: generate $x_i[0]$ randomly in the style of genetic programming and *gp_i* = 1);

/ receive from the user several values for initialization overwriting possibly the defaults. If atomic initialization fails re-initialize init. */*

2. **Goal** ($:=$ (*goal_i*[*t*] ($k\Omega_i[t]$ $x_i[t]$)) $<$ *goal_body* $>$): where *goal_body* checks for the maximum predefined quantum of time (to avoid undecidability or too long verification) whether

goal state defined in the init phase has been reached. If the quantum of time expires, it returns false \perp .

3. Select Phase

$(:= (sel\ x_i[t]) < (\sqcup\ cond_sel_exam\ (\circ\ \overline{cond_sel_exam}\ sel_body)) >):$ where $cond_sel_exam = (\sqcup\ (k_i = 0)\ (b_i = 0))$ and sel_body builds the search tree with the branching factor b_i and depth k_i over alphabet $A_i \cup \Omega_i$ starting from the current state $x_i[t]$. For each state s derive actions a being mulitsets of simple $\$$ -expressions, and arriving in a new state s' . Actions and new states are found in two ways:

- if gp_i flag is set to 1 - by applying crossover/mutation (in the form of send and receive operating on LTS trees) to obtain a new state s' . A corresponding action between s and s' will be labeled as observationally congruent ε with neutral cost 0.
- if gp_i flag is set to 0 - by applying inference rules of LTS to a state.

Each simple $\$$ -expression in actions is labeled

- by its name if simple $\$$ -expression belongs to $A_i \cup \Omega_i$ and width and depth b_i, k_i are not exceeded,
- is renamed by strongly congruent ε with estimated cost if flag $strongcong = 1$ (default: renamed by weakly congruent ε with a neutral (zero) cost, $strongcong = 0$) if width b_i or depth k_i are exceeded /* hiding actions outside of the agent's width or depth search horizon, however not ignoring, but estimating their costs */.

For each new state s' check whether width/depth of the tree is exceeded, and whether it is a goal state. If so, s' becomes the leaf of the tree (for the current loop cycle), and no new actions are generated, otherwise continue to build the tree. If s' is a goal state, label it as a goal state.

4. Examine Phase

$(:= (exam\ x_i[t]) < (\sqcup\ cond_sel_exam\ (\circ\ \overline{cond_sel_exam}\ exam_body)) >):$ where $exam_body$ prunes the search tree by selecting paths with minimal cost in cost choices and with maximal cost in adversary choices. Ties are broken randomly. In optimization, simple $\$$ -expressions belonging to $A_i - \Omega_i$ treat as observationally congruent ε with neutral cost (typically, equal to 0 like e.g., for a standard cost function) /* hiding agent's actions outside of its interests by ignoring their cost */.

5. Execute Phase

$(:= (exec\ (k\Omega_i[t]\ x_i[t]))\ exec_body):$ where $exec_body =$
 $(\circ\ (\sqcup\ (\circ\ (n_i = 0)(goal_reached)(current_node = leaf_node_with_min_costs))\ (\circ\ (n_i = 0)(goal_reached)(execute(x_i[t]))(current_node = leaf_node))$
 $(\circ\ (n_i = 0)(execute_n_i_steps(x_i[t]))$
 $(current_node = node_after_n_i_actions)))\ update_loop)$
 /* If $n_i = 0$ (offline search) and no goal state has been reached in the Select/Examine phase there will be no execution in this cycle. Pick up the most promising leaf node of the tree (with minimal cost) for expansion, i.e., make it a current node (root of the subtree expanded in the next cycle of the loop appended to an existing tree from the select phase, i.e., pruning will be invalidated to accommodate eventual corrections after cost updates). If $n_i = 0$ (offline search) and a goal state has been reached in the Select/Examine phase, execute optimal

$x_i[t]$ up to the leaf node using a tree constructed and pruned in the Select/Examine phase, or use LTS inference rules otherwise (for $gp = 1$). Make the leaf node a current node for a possible expansion (if it is not a goal state - it will be a root of a new tree) in the next cycle of the loop. If $n_i \neq 0$ (online search), execute optimal $x_i[t]$ up to at most n_i steps. Make the last state a current state - the root of the tree expanded in the next cycle of the loop. In execution simple $\$$ -expressions belonging to $\Omega_i - A_i$ will be executed by other agents.*/
The update_loop by default does nothing (executes silently ε with no cost) if update flag is reset. Otherwise if update = 1, then it gets from the user potentially all possible updates, e.g., new values of b_i , k_i , n_i and other parameters of $k\Omega[t]$, including costs of simple $\$$ -expressions, Ω_i , $goal_i$. If update = 1 and the user does not provide own modifications (including possible overwriting the $k\Omega[t]$), then self-modification will be performed in the following way. If execution was interrupted (by receiving message from the user or environment invalidating solution found in the Select/Examine phase), then $n_i = 10$ if $n_i = \infty$, or $n_i = n_i - 1$ if $n_i \neq 0$, or $k_i = 10$ if $n_i = 0$, $k_i = \infty$, or $k_i = k_i - 1$ if $n_i = 0$, $k_i \neq \infty$. If execution was not interrupted increase $n_i = n_i + 1$ pending $0 < n_i \leq k_i$. If $n_i = k_i$ increase $k_i = k_i + 1$, $b_i = b_i + 1$. If cost of search ($\$_{2i}(k\Omega[t])$) larger than a predefined threshold decrease k_i and/or b_i , otherwise increase it. If reinforcement learning was set up $rein = 1$ in the init phase, then cost of simple $\$$ -expressions will be modified by reinforcement learning.

The building of the LTS tree in the select phase for $gp_i = 0$ combines imperative and rule-based/logic styles of programming (we treat clause/production as a user-defined $\$$ -expression definition and call it by its name. This is similar to Robert Kowalski's dynamic interpretation of the left side of a clause as the name of procedure and the right side as the body of procedure.).

In the *init* and *exec/update* phase, in fact, a new search algorithm can be created (i.e., the old $k\Omega$ can be overwritten), and being completely different from the original $k\Omega$ -search. The original $k\Omega$ -search in self-modication changes the values of its control parameters mostly, i.e., k , n , b , but it could modify also *goal*, *sel*, *exam*, *exec* and $\$$.

Note that all parameters k_i , n_i , Ω_i , $\$i$, A_i , and \mathcal{A} can evolve in successive loop iterations. They are defined as the part of the *init* phase, and modified/updated at the end of the Execute phase *exec*. Note that they are associated with a specific choice of the meta-system: a $k\Omega$ -optimization search. For another meta-system, different control parameters are possible.

More details on the $k\Omega$ -search, including inference rules of the Labeled Transition System, observation and strong bisimulations and congruences, necessary and sufficient conditions to solve optimization, search optimization and total optimization problems, illustrating examples (including simulation by $k\Omega$ -optimization of A*, Minimax, Traveling Salesman Problem, and other typical search algorithms), the details of implementations and applications, can be found in (Eberbach, 2005a, 2007).

4. The $\$$ -Calculus Expressiveness and its Support to Solve TM Undecidable Problems

To deal with undecidability, the $\$$ -calculus uses all three principles from introductory section: the infinity, interaction, and evolution principles:

- *infinity* - because of the infinity of the indexing set I in the $\$$ -calculus operators, it is clear that the $\$$ -calculus derives its expressiveness mostly from the *infinity* principle.
- *interaction* - if to assume that simple $\$$ -expressions may represent oracles, then the $\$$ -calculus can represent the interaction principle. Then we define an equivalent of the oracle as a user defined simple $\$$ -expression, that somehow in the manner of the “black-box” solves unsolvable problems (however, we do not know how).
- *evolution* - the $k\Omega$ -optimization may be evolved to a new (and hopefully) more powerful problem solving method

It is easier and “cleaner” to think about implementation of unbounded (infinitary) concepts, than about implementation of oracles. The implementation of scalable computers (e.g., scalable massively parallel computers or unbounded growth of Internet) allows to think about a reasonable approximation of the implementation of *infinity* (and, in particular, the π -calculus, or the $\$$ -calculus). At this point, it is not clear how to implement oracles (as Turing stated *an oracle cannot be a machine*, i.e., implementable by mechanical means), and as the result, the models based on them. One of potential implementation of oracles could be an infinite lookup table with stored all results of the decision for TM. The quite different story is how to initialize such infinite lookup table and how to search it effectively using for instance hash indices or B+ trees.

The expressiveness of the $\$$ -calculus is not worse than the expressiveness of Turing Machines, i.e., it is straightforward to show how to encode λ -calculus (Church, 1936, 1941) in $\$$ -calculus (Eberbach, 2006, 2007). In (Eberbach, 2005a) it has been demonstrated, how some other models of computation, more expressive than Turing Machines, can be simulated in the $\$$ -calculus. This includes the π -calculus, Interaction Machines, cellular automata, neural networks, and random automata networks.

It is interesting that the $\$$ -calculus can solve in the limit the halting problem of the Universal Turing Machine, and approximate the solution of the halting/optimization problem of the $\$$ -calculus. This is a very interesting result, because, if correct, besides the $\$$ -calculus, it may suggest that a self-modifying program using infinitary means may approximate the solution of its own decision (halting or optimization) problem.

4.1. Solving the Turing Machine Halting Problem and Approximating the Universal Search Algorithm

The results from Eberbach (2006, 2007) justify that the $\$$ -calculus is more expressive than the TM, and may represent non-algorithmic computation. In Eberbach (2006, 2007) three ways how the $\$$ -calculus solves the halting problem of the Universal Turing Machine using either an *infinity*, *evolution*, or *interaction principle*.

Theorem 4.1 (On solution of the halting problem of UTM by $\$$ -calculus (Eberbach, 2006, 2007)). *The halting problem for the Universal Turing Machine is solvable by the $\$$ -calculus.*

Proof. (Outline): In the infinity principle $\$$ -calculus taking an instance of TM code and its input runs an infinite number of steps of TM (same idea like for example in Infinite Time TM). Of course, in infinity the answer for halting will be yes or no. It is a matter of philosophical discussion whether

getting a definitive answer in infinity constitutes an answer at all. In the interaction principle, $\$$ -calculus $k\Omega$ -search gets an answer from oracle, and in the evolution principle, the TM is evolved to TM with oracle. Assuming that a TM with an oracle can be encoded as a nonrecursive function ($\$$ -expression) using a binary alphabet and an ordinary TM can be (of course) encoded as a binary string, thus a simple binary mutation changing one binary input to another can convert by chance an ordinary Turing Machine to the Turing Machine with an oracle (Turing, 1939). The probability of such event is very small, and because nobody has implemented the Turing Machine with an oracle (although we know what its transition function may look like - see, e.g. (Kozen, 1997)). We may have the big problem with the recognition of this encoding, thus even if theoretically possible, so far nobody has been able to detect the potential existence of the Turing Machine with an oracle. \square

We know from the theory of computation that the search for the universal algorithm is a futile effort. If it was not so, such algorithm could be used immediately to solve the halting problem of TM.

We will show how the $\$$ -calculus can help potentially for the solution of the best search algorithm by approximating it. The best search algorithm will be in the sense of the optimum: finding the best-quality solutions/search algorithms for the all possible problems. The trouble and undecidability is caused by the requirement to cover exactly *all* possible problems. The number of possible algorithms is enumerable, however, the number of possible problems is infinite, but not enumerable (problems/languages are all possible subsets of all algorithms), see, e.g., (Hopcroft et al., 2001).

Theorem 4.2 (On approximating the universal search algorithm (Eberbach, 2006, 2007)). *The $k\Omega$ -optimization taking itself as its input will converge with an arbitrarily small error in finite time to the universal search algorithm if search is complete and elitist selection strategy is used.*

Proof. (Outline): In the above approach completeness guarantees that no solution will be missed and elitist selection allows to preserve the best solution found so far. In other words, the $k\Omega$ -search taking as its input itself, produces in the finite time better versions of itself and in infinity reaches the optimum (pending that it exists). This allows to approximate the best search algorithm in the finite time. In particular, this can be used for approximated solution of the UTM halting problem. \square

In such way progress in mathematics or computer science (both being undecidable) is done. Proving all theorems (existing and not derived yet) in mathematics or computer science is impossible (the Entscheidungsproblem - Hilbert's decision problem is undecidable (Whitehead & Russell, 1910, 1912, 1913; Turing, 1937; Hopcroft et al., 2001)). However, in spite of that, new generations of mathematicians and computer scientists work and generate new theorems and improve indefinitely the current state of art of mathematics and computer science. The famous unsolvable Entscheidungsproblem does not prevent scientists from discovering new theorems, improving our knowledge of mathematics, but we will never be able to write all theorems (unless to wait for eternity).

4.2. Deciding the Diagonalization Language, Nontrivial Properties, Solving Post Correspondence Problem and Busy Beaver Problem

In (Eberbach, 2003) several open problems have been posed. We will present solutions to some of them.

The diagonalization language L_d is an example of the language that is believed even tougher than L_u , i.e., language of UTM accepting words w for arbitrary TM M . L_d is non-RE, i.e., it does not exist any TM accepting it.

Definition 4.1. The diagonalization language L_d consists of all strings w such that TM M whose code is w does not accept when given w as input.

The existence of diagonalization language that cannot be accepted by any TM is proven by diagonalization table with “dummy”, i.e., not real/true values. Of course, there are many diagonalization language encodings possible that depend how transitions of TMs are encoded. This means that there are infinitely many different L_d language instances (but nobody wrote a specific example of L_d). Solving the halting problem of UTM, can be used for a “constructive” proof of the diagonalization language decidability demonstrating all strings belonging to the language.

Theorem 4.3 (Deciding asymptotically the diagonalization language). *The diagonalization language L_d is $\$$ -calculus asymptotically decidable pending that language of halting UTM L_u is decidable.*

Proof. By Theorem 4.1 we fill a characteristic vector for each TM in diagonalization table, i.e., for each TM M_i and for each its input string w_j , $i, j = 0, 1, 2, \dots$, we write 1 if w_j is accepted by M_i and 0 otherwise. This is always possible pending that halting of UTM is solvable. For each M_i we look at w_i (the diagonal value) and flip its bit to opposite value. Now for each w_i , $i = 0, 1, 2, \dots$, we construct a characteristic vector for L_d . For each accepted string w_i we construct a finite automaton FA_i accepting exactly one word w_i (always trivially possible in finite time). We construct an infinite parallel composition - $\$$ -calculus $\$$ -expression $(\parallel_i FA_i)$ and put to it an arbitrary input string w . If $(\parallel_i FA_i)$ accepts w (i.e., one of FA accepts) then L_d accepts. If no FA accepts then L_d does not accept either. \square

In such a way we can decide in $\$$ -calculus a language L_d that is not possible to decide in the TM model.

In an analogous way we can decide other TM unsolvable languages/problems.

The language L_{ne} consisting of all binary encoded TMs whose language is not empty, i.e., $L_{ne} = \{M \mid L(M) \neq \emptyset\}$ is known to be recursively enumerable but not recursive, and its complement language $L_e = \{M \mid L(M) = \emptyset\}$ consisting of all binary encoded TMs whose language is empty is known to be non recursively enumerable.

Theorem 4.4 (Deciding asymptotically L_{ne} and L_e). *The languages L_{ne} and L_e are $\$$ -calculus asymptotically decidable pending that language of halting UTM L_u is decidable.*

Proof. It is enough to look at the diagonalization table found after solving UTM L_u halting problem. The rows containing at least one 1 allow to decide L_{ne} , and complementary rows consisting of all 0s allow to decide L_e . \square

We can solve nontrivial properties (nonempty proper subsets of all RE languages), i.e., to prove solvability of Rice theorem (Rice, 1953) using hypercomputation.

Theorem 4.5 (Deciding asymptotically nontrivial properties). *Every nontrivial property is Σ -calculus asymptotically decidable pending that language of halting UTM L_u is decidable.*

Proof. We identify the proper subset of the rows from the diagonalization table satisfying a specific nonempty property. It is necessary to translate specific property in terms of corresponding 1s and 0s in characteristic vectors for each TM M_i (e.g., to translate which rows correspond to the empty language, a finite language, a regular language, context-free language, context-sensitive language). \square

It is also possible to decide PCP and Busy Beaver Problem.

Definition 4.2. The TM undecidable Post Correspondence Problem (PCP) (Post, 1946) asks, given two lists of the same number of strings over the same alphabet, whether we can pick a sequence of corresponding strings from the two lists and form the same string by concatenation.

Theorem 4.6 (Deciding asymptotically PCP). *The Post Correspondence Problem is asymptotically decidable pending that language of halting UTM L_u is decidable.*

Proof. As the halting or terminal state of the TM solving PCP we put the condition whether both lists produce the same string. \square

Definition 4.3. The TM undecidable Busy Beaver Problem (BBP) (Rado, 1962) considers a deterministic 1-tape Turing machine with unary alphabet $\{1\}$ and tape alphabet $\{1, B\}$, where B represents the tape blank symbol. TM starts with an initial empty tape and accepts by halting. For the arbitrary number of states $n = 0, 1, 2, \dots$ TM tries to compute two functions: the maximum number of 1s written on tape before halting (known as the busy beaver function $\Sigma(n)$) and the maximum number of steps before halting (known as the maximum shift function $S(n)$).

Theorem 4.7 (Deciding asymptotically BBP). *The Busy Beaver Problem is asymptotically decidable pending that language of halting UTM L_u is decidable.*

Proof. As the halting state of the TM solving BBP we put the condition whether $\Sigma(n)$ and $S(n)$ have been computed. \square

5. Conclusions

In the paper some hypercomputation solutions of Turing Machine unsolvable problems have been presented. We demonstrate that the solution of the halting problem is like to solve polynomially one NP-complete problem to resolve famous dilemma $\mathcal{P} \neq \mathcal{NP}$, i.e., it breaks the whole hierarchical puzzle of unsolvability. Namely, solving the halting problem of UTM is pivotal to solve many other Turing Machine unsolvable problems, including to decide the diagonalization language, nontrivial properties, PCP and BBP.

However, hypercomputational models are still not well researched and many scientists vigorously oppose the idea that computations going beyond Turing Machine are possible at all. Very

little is known about the hierarchy (or at least relations) between hypercomputational models. We do not know enough about the implementability of hypercomputation at least at the level similar to quantum computing or biomolecular computing implementations. We do not know the limits of computability in hypercomputational models. We do not know whether such limits exist at all. In other words, hypercomputation is in the situation similar like Turing, Church and Gödel were in 1930s before the whole digital computers explosive growth started. Whether these hopes and fears about hypercomputation will be materialized is a quite different story.

References

- Aho, Alfred V. (2011). Ubiquity symposium: Computation and computational thinking. *Ubiquity*.
- Burgin, M. S. (2004). *Super-Recursive Algorithms (Monographs in Computer Science)*. SpringerVerlag.
- Church, A. (1936). An unsolvable problem of elementary number theory. *American Journal of Mathematics* **58**(2), 345–363.
- Church, A. (1941). *The Calculi of Lambda Conversion*. Princeton, N.J., Princeton University Press. New York, NY, USA.
- Cooper, B. (2012). Turing’s titanic machine?. *Commun. ACM* **55**(3), 74–83.
- Eberbach, E. (1997). A generic tool for distributed AI with matching as message passing. In: *Tools with Artificial Intelligence, 1997. Proceedings., Ninth IEEE International Conference on*. pp. 11–18.
- Eberbach, E. (2003). Is entscheidungsproblem solvable? beyond undecidability of Turing machines and its consequence for computer science and mathematics. In: (ed. J. C. Misra) *Computational Mathematics, Modelling and Algorithms*. pp. 1–32. Narosa Publishing House, New Delhi.
- Eberbach, E. (2005a). $\$$ -Calculus of bounded rational agents: Flexible optimization as search under bounded resources in interactive systems. *Fundam. Inf.* **68**(1-2), 47–102.
- Eberbach, E. (2005b). Toward a theory of evolutionary computation. *Biosystems* **82**(1), 1 – 19.
- Eberbach, E. (2006). Expressiveness of the π -Calculus and the $\$$ -Calculus. In: *Proc. 2006 World Congress in Comp. Sci., Comp. Eng., & Applied Computing, The 2006 Intern. Conf. on Foundations of Computer Science FCS’06, Las Vegas, Nevada*. pp. 24–30.
- Eberbach, E. (2007). The $\$$ -calculus process algebra for problem solving: A paradigmatic shift in handling hard computational problems. *Theoretical Computer Science* **383**(23), 200 – 243. Complexity of Algorithms and Computations.
- Eberbach, E. and M. Burgin (2009). On foundations of evolutionary computation: An evolutionary automata approach. In: (ed. Hongwei Mo): *Handbook of Research on Artificial Immune Systems and Natural Computing: Applying Complex Adaptive Technologies, Section II: Natural Computing, Section II.1: Evolutionary Computing, Chapter XVI, Medical Information Science Reference/IGI Global, Hershey*. pp. 342–360. New York.
- Eberbach, E., D. Goldin and P. Wegner (2004). Turings ideas and models of computation. In: *Alan Turing: Life and Legacy of a Great Thinker* (Christof Teuscher, Ed.). pp. 159–194. Springer Berlin Heidelberg.
- Eberbach, Eugene and Peter Wegner (2003). Beyond Turing machines. *Bulletin of the EATCS* pp. 279–304.
- Hopcroft, J. E., R. Motwani and J. D. Ullman (2001). Introduction to automata theory, languages, and computation, 2nd edition. *SIGACT News* **32**(1), 60–65.
- Horvitz, E. and S. Zilberstein (2001). Computational tradeoffs under bounded resources. *Artificial Intelligence* **126**(12), 1 – 4. Tradeoffs under Bounded Resources.
- Kozen, D. C. (1997). *Automata and Computability*. Springer-Verlag.
- Milner, R. (1999). *Communicating and Mobile Systems: The π -calculus*. Cambridge University Press. New York, NY, USA.

- Milner, R., J. Parrow and D. Walker (1992). A calculus of mobile processes, i. *Information and Computation* **100**(1), 1 – 40.
- Post, E. (1946). A variant of a recursively unsolvable problem. *Bull. Amer. Math. Soc.* **52**(4), 264–268.
- Rado, T. (1962). On non-computable functions. *Bell System Technical Journal* **41**(3), 877–884.
- Rice, H. G. (1953). Classes of recursively enumerable sets and their decision problems. *Trans. Amer. Math. Soc.* **74**, 358–366.
- Russell, S. and P. Norvig (2002). *Artificial Intelligence: A Modern Approach (2nd Edition)*. Prentice Hall.
- Syropoulos, A. (2007). *Hypercomputation: Computing Beyond the Church-Turing Barrier (Monographs in Computer Science)*. Springer-Verlag New York, Inc.. Secaucus, NJ, USA.
- Turing, A. M. (1937). On computable numbers, with an application to the entscheidungsproblem. *Proceedings of the London Mathematical Society* **s2-42**(1), 230–265.
- Turing, A. M. (1939). Systems of logic based on ordinals. *Proceedings of the London Mathematical Society* **s2-45**(1), 161–228.
- Wegner, P., E. Eberbach and M. Burgin (2012). Computational completeness of interaction machines and Turing machines. In: *Turing-100* (Andrei Voronkov, Ed.). Vol. 10 of *EPiC Series*. EasyChair. pp. 405–414.
- Whitehead, A. N. and B. Russell (1910, 1912, 1913). *Principia mathematica*, vol.1, 1910, vol.2, 1912, vol.3, 1913. Cambridge Univ. Press. Cambridge.



Coupled Systems of Fractional Integro-Differential Equations Involving Several Functions

Zoubir Dahmani^{a,*}, Mohamed Amin Abdellaoui^a, Mohamed Houas^b

^aLaboratory LPAM, Faculty SEI, UMAB University, Algeria

^bLaboratory FIMA, Faculty ST, University of Khemis Miliana, Algeria

Abstract

This paper studies the existence of solutions for a coupled system of nonlinear fractional integro-differential equations involving Riemann-Liouville integrals with several continuous functions. New existence and uniqueness results are established using Banach fixed point theorem, and other existence results are obtained using Schaefer fixed point theorem. Some illustrative examples are also presented.

Keywords: Caputo derivative, fixed point, integro-differential system, existence, uniqueness, Riemann-Liouville integral.

2010 MSC: 34A34, 34B10.

1. Introduction

The differential equations of fractional order arise in many scientific disciplines, such as physics, chemistry, control theory, signal processing and biophysics. For more details, we refer the reader to (Kilbas & Marzan, 2005; Lakshmikantham & Vatsala, 2008; Su, 2009) and the references therein. Recently, there has been a significant progress in the investigation of these equations, (see (Anber *et al.*, 2013; Bengrine & Dahmani, 2012; Cui *et al.*, 2012; Wang *et al.*, 2010; Zhang, 2006)). On the other hand, the study of coupled systems of fractional differential equations is also of a great importance. Such systems occur in various problems of applied science. For some recent results on the fractional systems, we refer the reader to (Abdellaoui *et al.*, 2013; Bai & Fang, 2004; Gaber & Brikaa, 2012; Gafiychuk *et al.*, 2008).

*Corresponding author

Email addresses: zzdahmani@yahoo.fr (Zoubir Dahmani), abdellaouiamine13@yahoo.fr (Mohamed Amin Abdellaoui), houasmed@yahoo.fr (Mohamed Houas)

In this paper, we discuss the existence and uniqueness of solutions for the following coupled system of fractional integro-differential equations:

$$\begin{cases} D^\alpha u(t) = f_1(t, u(t), v(t)) + \sum_{i=1}^m \int_0^t \frac{(t-s)^{\alpha_i-1}}{\Gamma(\alpha_i)} \varphi_i(s) g_i(s, u(s), v(s)) ds, \\ D^\beta v(t) = f_2(t, u(t), v(t)) + \sum_{i=1}^m \int_0^t \frac{(t-s)^{\beta_i-1}}{\Gamma(\beta_i)} \phi_i(s) h_i(s, u(s), v(s)) ds, \\ u(0) = a > 0, v(0) = b > 0, t \in [0, 1] \end{cases} \quad (1.1)$$

where D^α, D^β denote the Caputo fractional derivatives, $0 < \alpha < 1, 0 < \beta < 1, \alpha_i, \beta_i$ are non negative real numbers, φ_i and ϕ_i are continuous functions, $m \in \mathbb{N}^*$, f_1, f_2 and g_i and $h_i, i = 1, \dots, m$, are functions that will be specified later.

The paper is organized as follows: In section 2, we present some preliminaries and lemmas. Section 3 is devoted to existence of solutions of problem (1.1). In the last section, some examples are presented to illustrate our results.

2. Preliminaries

The following notations, definitions and lemmas will be used throughout this paper.

Definition 2.1. The Riemann-Liouville fractional integral operator of order $\alpha > 0$, for $f \in L^1([a, b], \mathbb{R})$ is defined by:

$$I^\alpha f(t) = \int_a^t \frac{(t-\tau)^{\alpha-1}}{\Gamma(\alpha)} f(\tau) d\tau, \quad a \leq t \leq b, \quad (2.1)$$

where $\Gamma(\alpha) := \int_0^\infty e^{-u} u^{\alpha-1} du$.

Definition 2.2. The fractional derivative of $f \in C^n([a, b], \mathbb{R})$, $n \in \mathbb{N}^*$, in the sense of Caputo, of order α , $n-1 < \alpha < n$ is defined by:

$$D^\alpha f(t) = \int_a^t \frac{(t-\tau)^{n-\alpha-1}}{\Gamma(n-\alpha)} f^{(n)}(\tau) d\tau, \quad t \in [a, b]. \quad (2.2)$$

For more details about fractional calculus, we refer the reader to (Mainardi, 1997). The following lemmas give some properties of Riemann-Liouville integrals and Caputo fractional derivatives (Kilbas & Marzan, 2005; Lakshmikantham & Vatsala, 2008):

Lemma 2.1. Given $f \in L^1([a, b], \mathbb{R})$, then for all $t \in [a, b]$ we have $I^r I^s f(t) = I^{r+s} f(t)$, for $r, s > 0$. $D^s I^s f(t) = f(t)$, for $s > 0$. $D^r I^s f(t) = I^{s-r} f(t)$, for $s > r > 0$.

To study the coupled system (1.1), we need the following two lemmas (Kilbas & Marzan, 2005):

Lemma 2.2. For $n - 1 < \alpha < n$, where $n \in \mathbb{N}^*$, the general solution of the equation $D^\alpha x(t) = 0$ is given by

$$x(t) = c_0 + c_1 t + c_2 t^2 + \dots + c_{n-1} t^{n-1}, \quad (2.3)$$

where $c_i \in \mathbb{R}, i = 0, 1, 2, \dots, n - 1$.

Lemma 2.3. Let $n - 1 < \alpha < n$, where $n \in \mathbb{N}^*$. Then, for $x \in C^n([0, 1], \mathbb{R})$, we have

$$I^\alpha D^\alpha x(t) = x(t) + c_0 + c_1 t + c_2 t^2 + \dots + c_{n-1} t^{n-1}, \quad (2.4)$$

for some $c_i \in \mathbb{R}, i = 0, 1, 2, \dots, n - 1, n = [\alpha] + 1$.

We prove the following auxiliary lemma:

Lemma 2.4. Let $f, R_i, K_i \in C([0, 1], \mathbb{R}), i = 1, \dots, m$. The solution of the problem

$$D^\alpha x(t) = f(t) + \sum_{i=1}^m \int_0^t \frac{(t-s)^{\alpha_i-1}}{\Gamma(\alpha_i)} R_i(s) K_i(s) ds, 0 < \alpha < 1, \quad \alpha_i > 0 \quad (2.5)$$

with the condition, $x(0) = x_0^* \in \mathbb{R}_+^*$, is given by

$$x(t) = \int_0^t \frac{(t-s)^{\alpha-1}}{\Gamma(\alpha)} f(s) ds + \sum_{i=1}^m \int_0^t \frac{(t-s)^{\alpha+\alpha_i-1}}{\Gamma(\alpha+\alpha_i)} R_i(s) K_i(s) ds + x_0^*. \quad (2.6)$$

Proof. Setting

$$y(t) = x(t) - I^\alpha f(t) - \sum_{i=1}^m I^{\alpha+\alpha_i} R_i(t) K_i(t). \quad (2.7)$$

Thanks to the linearity of D^α , we get

$$D^\alpha y(t) = D^\alpha x(t) - D^\alpha I^\alpha f(t) - \sum_{i=1}^m D^\alpha I^{\alpha+\alpha_i} R_i(t) K_i(t). \quad (2.8)$$

By lemma 2.2, yields

$$D^\alpha y(t) = D^\alpha x(t) - f(t) - \sum_{i=1}^m I^{\alpha_i} R_i(t) K_i(t). \quad (2.9)$$

Thus, (2.5) is equivalent to $D^\alpha y(t) = 0$.

Finally, thanks to lemma 2.3, we obtain that $y(t)$ is constant, i.e., $y(t) = y(0) = x(0) = x_0^*$, and the proof of lemma 2.4 is achieved. \square

3. Main Results

We introduce in this paragraph the following assumptions:

(H1) : There exist non negative real numbers $\mu_j, \nu_j, j = 1, 2$ and $l_i, m_i, n_i, k_i, i = 1, \dots, m$, such that for all $t \in [0, 1]$ and $(u_1, v_1), (u_2, v_2) \in \mathbb{R}^2$, we have

$$|f_j(t, u_2, v_2) - f_j(t, u_1, v_1)| \leq \mu_j |u_2 - u_1| + \nu_j |v_2 - v_1|, j = 1, 2$$

$$|g_i(t, u_2, v_2) - g_i(t, u_1, v_1)| \leq l_i |u_2 - u_1| + m_i |v_2 - v_1|, i = 1, \dots, m,$$

and,

$$|h_i(t, u_2, v_2) - h_i(t, u_1, v_1)| \leq n_i |u_2 - u_1| + k_i |v_2 - v_1|, i = 1, \dots, m$$

with

$$\bar{L} = \max(\mu_1, \mu_2, \nu_1, \nu_2, l_i, m_i, n_i, k_i, i = 1, \dots, m.)$$

(H2) : The functions f_1, f_2, g_i and $h_i : [0, 1] \times \mathbb{R}^2 \rightarrow \mathbb{R}$ are continuous for each $i = 1, \dots, m$.

(H3) : There exist positive real numbers $L_1, L_2, L'_i, L''_i, i = 1, \dots, m$, such that

$$\begin{aligned} |f_1(t, u, v)| &\leq L_1, |g_i(t, u, v)| \leq L'_i, |f_2(t, u, v)| \leq L_2, \\ |h_i(t, u, v)| &\leq L''_i, t \in [0, 1], (u, v) \in \mathbb{R}^2. \end{aligned}$$

Our first result is given by:

Theorem 3.1. Assume that (H1) holds and setting

$$\begin{aligned} M_1 &:= \frac{1}{\Gamma(\alpha + 1)} + \sum_{i=1}^m \frac{\|\varphi_i\|_\infty}{\Gamma(\alpha + \alpha_i + 1)}, \\ M_2 &:= \frac{1}{\Gamma(\beta + 1)} + \sum_{i=1}^m \frac{\|\phi_i\|_\infty}{\Gamma(\beta + \beta_i + 1)}. \end{aligned}$$

If

$$\bar{L}(M_1 + M_2) < 1, \quad (3.1)$$

then, the system (1.1) has exactly one solution on $[0, 1]$.

Proof. Setting $X := C([0, 1], \mathbb{R})$. This space, equipped with the norm $\|\cdot\|_X = \|\cdot\|_\infty$ defined by $\|f\|_\infty = \sup\{|f(x)|, x \in [0, 1]\}$, is a Banach space. Also, the product space $(X \times X, \|(u, v)\|_{X \times X})$ is a Banach space, with $\|(u, v)\|_{X \times X} = \|u\|_X + \|v\|_X$.

Consider now the operator $\Psi : X \times X \rightarrow X \times X$ defined by

$$\Psi(u, v)(t) = \left(\Psi_1(u, v)(t), \Psi_2(u, v)(t) \right), \quad (3.2)$$

where

$$\Psi_1(u, v)(t) = \int_0^t \frac{(t-s)^{\alpha-1}}{\Gamma(\alpha)} f_1(s, u(s), v(s)) ds + \sum_{i=1}^m \int_0^t \frac{(t-s)^{\alpha+\alpha_i-1}}{\Gamma(\alpha+\alpha_i)} \varphi_i(s) g_i(s, u(s), v(s)) ds + a. \quad (3.3)$$

and

$$\Psi_2(u, v)(t) = \int_0^t \frac{(t-s)^{\beta-1}}{\Gamma(\beta)} f_2(s, u(s), v(s)) ds + \sum_{i=1}^m \int_0^t \frac{(t-s)^{\beta+\beta_i-1}}{\Gamma(\beta+\beta_i)} \varphi_i(s) g_i(s, u(s), v(s)) ds + b. \quad (3.4)$$

We shall show that Ψ is contractive: Let $(u_1, v_1), (u_2, v_2) \in X \times X$. Then, for each $t \in [0, 1]$, we have

$$\begin{aligned} |\Psi_1(u_2, v_2)(t) - \Psi_1(u_1, v_1)(t)| \leq & \int_0^t \frac{(t-s)^{\alpha-1}}{\Gamma(\alpha)} ds \times \sup_{0 \leq s \leq 1} |f_1(s, u_2(s), v_2(s)) - f_1(s, u_1(s), v_1(s))| \\ & + \sum_{i=1}^m \left(\sup_{0 \leq s \leq 1} |\varphi_i(s)| \int_0^t \frac{(t-s)^{\alpha+\alpha_i-1}}{\Gamma(\alpha+\alpha_i)} ds \right. \\ & \left. \times \sup_{0 \leq s \leq 1} |g_i(s, u_2(s), v_2(s)) - g_i(s, u_1(s), v_1(s))| \right). \end{aligned} \quad (3.5)$$

Therefore,

$$\begin{aligned} |\Psi_1(u_2, v_2)(t) - \Psi_1(u_1, v_1)(t)| \leq & \frac{1}{\Gamma(\alpha+1)} \sup_{0 \leq s \leq 1} |f_1(s, u_2(s), v_2(s)) - f_1(s, u_1(s), v_1(s))| \\ & + \sum_{i=1}^m \frac{\|\varphi_i\|_\infty}{\Gamma(\alpha+\alpha_i+1)} \sup_{0 \leq s \leq 1} |g_i(s, u_2(s), v_2(s)) - g_i(s, u_1(s), v_1(s))|. \end{aligned} \quad (3.6)$$

Using (H1), we can write:

$$\begin{aligned} |\Psi_1(u_2, v_2)(t) - \Psi_1(u_1, v_1)(t)| \leq & \frac{\bar{L}}{\Gamma(\alpha+1)} \left(\sup_{0 \leq t \leq 1} |u_2(t) - u_1(t)| + \sup_{0 \leq t \leq 1} |v_2(t) - v_1(t)| \right) \\ & + \sum_{i=1}^m \frac{\|\varphi_i\|_\infty \bar{L}}{\Gamma(\alpha+\alpha_i+1)} \left(\sup_{0 \leq t \leq 1} |u_2(t) - u_1(t)| + \sup_{0 \leq t \leq 1} |v_2(t) - v_1(t)| \right). \end{aligned} \quad (3.7)$$

This implies that

$$|\Psi_1(u_2, v_2)(t) - \Psi_1(u_1, v_1)(t)| \leq M_1 \bar{L} (\|u_2 - u_1\|_X + \|v_2 - v_1\|_X). \quad (3.8)$$

And consequently,

$$\|\Psi_1(u_2, v_2) - \Psi_1(u_1, v_1)\|_X \leq M_1 \bar{L} \|(u_2 - u_1, v_2 - v_1)\|_{X \times X}. \quad (3.9)$$

With the same arguments as before, we can write

$$\|\Psi_2(u_2, v_2) - \Psi_2(u_1, v_1)\|_X \leq M_2 \bar{L} \|(u_2 - u_1, v_2 - v_1)\|_{X \times X}. \quad (3.10)$$

Finally, using (3.9) and (3.10), we deduce that

$$\|\Psi(u_2, v_2) - \Psi(u_1, v_1)\|_{X \times X} \leq \bar{L}(M_1 + M_2) \|(u_2 - u_1, v_2 - v_1)\|_{X \times X}. \quad (3.11)$$

Thanks to (3.1), we conclude that Ψ is a contraction mapping. Hence, by Banach fixed point theorem, there exists a unique fixed point which is a solution of (1.1). \square

The second result is the following:

Theorem 3.2. Assume that (H2) and (H3) are satisfied and $L'_i \leq L_1, L''_i \leq L_2, i = 1, \dots, m$. Then problem (1.1) has at least one solution on $[0, 1]$.

Proof. First of all, we show that the operator T is completely continuous.

Step 1: Let us take $\gamma > 0$ and $B_\gamma := \{(u, v) \in X \times X; \|(u, v)\|_{X \times X} \leq \gamma\}$. Now, assume that (H3) holds, and $L'_i \leq L_1, L''_i \leq L_2$. Then for $(u, v) \in B_\gamma$, we have

$$\begin{aligned} |\Psi_1(u, v)(t)| &\leq a + \frac{t^\alpha}{\Gamma(\alpha+1)} \sup_{0 \leq t \leq 1} |f_1(t, u(t), v(t))| \\ &\quad + \sum_{i=1}^m \frac{\|\varphi_i\|_\infty t^{\alpha+\alpha_i}}{\Gamma(\alpha+\alpha_i+1)} \sup_{0 \leq t \leq 1} |g_i(t, u(t), v(t))|, t \in [0, 1]. \end{aligned} \quad (3.12)$$

Hence, we obtain

$$\|\Psi_1(u, v)\|_X \leq L_1 M_1 + a < +\infty. \quad (3.13)$$

With the same arguments, we have

$$\|\Psi_2(u, v)\|_X \leq L_2 M_2 + b < +\infty. \quad (3.14)$$

Then, by (23) and (24), we can state that $\|T(u, v)\|_{X \times X}$ is bounded by C , where

$$C := L_1 M_1 + L_2 M_2 + a + b. \quad (3.15)$$

Step 2: Let $t_1, t_2 \in [0, 1], t_1 < t_2$ and $(u, v) \in B_\gamma$. We have

$$\begin{aligned} |\Psi_1(u, v)(t_2) - \Psi_1(u, v)(t_1)| &\leq \left| \int_0^{t_2} \frac{(t_2-s)^{\alpha-1}}{\Gamma(\alpha)} f_1(s, u(s), v(s)) ds - \int_0^{t_1} \frac{(t_1-s)^{\alpha-1}}{\Gamma(\alpha)} f_1(s, u(s), v(s)) ds \right| \\ &\quad + \left| \sum_{i=1}^m \int_0^{t_2} \frac{(t_2-s)^{\alpha+\alpha_i-1}}{\Gamma(\alpha+\alpha_i)} \varphi_i(s) g_i(s, u(s), v(s)) ds - \sum_{i=1}^m \int_0^{t_1} \frac{(t_1-s)^{\alpha+\alpha_i-1}}{\Gamma(\alpha+\alpha_i)} \varphi_i(s) g_i(s, u(s), v(s)) ds \right|. \end{aligned} \quad (3.16)$$

Thus, we get

$$|\Psi_1(u, v)(t_2) - \Psi_1(u, v)(t_1)| \leq \frac{L_1(t_2^\alpha - t_1^\alpha + (t_2 - t_1)^\alpha)}{\Gamma(\alpha+1)} + \sum_{i=1}^m \frac{L_1 \|\varphi_i\|_\infty (t_2^{\alpha+\alpha_i} - t_1^{\alpha+\alpha_i} + (t_2 - t_1)^{\alpha+\alpha_i})}{\Gamma(\alpha+\alpha_i+1)}. \quad (3.17)$$

Analogously, we can obtain

$$|\Psi_2(u, v)(t_2) - \Psi_2(u, v)(t_1)| \leq \frac{L_2(t_2^\beta - t_1^\beta + (t_2 - t_1)^\beta)}{\Gamma(\beta + 1)} + \sum_{i=1}^m \frac{L_2 \|\varphi_i\|_\infty (t_2^{\beta+\beta_i} - t_1^{\beta+\beta_i} + (t_2 - t_1)^{\beta+\beta_i})}{\Gamma(\beta + \beta_i + 1)}. \quad (3.18)$$

Therefore,

$$\begin{aligned} |\Psi(u, v)(t_2) - \Psi(u, v)(t_1)| &\leq \frac{L_1(t_2^\alpha - t_1^\alpha + (t_2 - t_1)^\alpha)}{\Gamma(\alpha + 1)} + \sum_{i=1}^m \frac{L_1 \|\varphi_i\|_\infty (t_2^{\alpha+\alpha_i} - t_1^{\alpha+\alpha_i} + (t_2 - t_1)^{\alpha+\alpha_i})}{\Gamma(\alpha + \alpha_i + 1)} \\ &\quad + \frac{L_2(t_2^\beta - t_1^\beta + (t_2 - t_1)^\beta)}{\Gamma(\beta + 1)} + \sum_{i=1}^m \frac{L_2 \|\varphi_i\|_\infty (t_2^{\beta+\beta_i} - t_1^{\beta+\beta_i} + (t_2 - t_1)^{\beta+\beta_i})}{\Gamma(\beta + \beta_i + 1)}. \end{aligned} \quad (3.19)$$

As $t_2 \rightarrow t_1$, the right-hand side of (3.19) tends to zero. Then, as a consequence of Steps 1, 2, and by Arzela-Ascoli theorem, we conclude that Ψ is completely continuous.

Next, we consider the set:

$$\Omega = \{(u, v) \in X \times X; (u, v) = \lambda T(u, v), 0 < \lambda < 1\}. \quad (3.20)$$

We shall show that Ω is bounded:

Let $(u, v) \in \Omega$, then $(u, v) = \lambda \Psi(u, v)$, for some $0 < \lambda < 1$. Hence, for $t \in [0, 1]$, we have:

$$u(t) = \lambda \Psi_1(u, v)(t), v(t) = \lambda \Psi_2(u, v)(t). \quad (3.21)$$

Thus,

$$\|(u, v)\|_{X \times X} = \lambda \|\Psi(u, v)\|_{X \times X}. \quad (3.22)$$

Thanks to (H_3) ,

$$\|(u, v)\|_{X \times X} \leq \lambda C, \quad (3.23)$$

where C is defined by (3.15). Therefore, Ω is bounded.

As a conclusion of Schaefer fixed point theorem, we deduce that Ψ has at least one fixed point, which is a solution of (1.1). \square

4. Examples

Example 4.1. Consider the following fractional system:

$$\begin{cases} D^{\frac{1}{2}} u(t) = \left(\frac{\sin(u(t)+v(t))}{18(\ln(t+1)+1)} + 6 \right) + \int_0^t \frac{(t-s)^{\frac{1}{2}}}{\Gamma(\frac{3}{2})} \left(\frac{\exp(-s)}{18(s+1)} \frac{\sin(u(s)+v(s))}{18(s+5)} \right) ds, t \in [0, 1], \\ D^{\frac{1}{2}} v(t) = \frac{\sin u(s)+\sin v(s)}{16(t \exp(t^2)+1)} + \int_0^t \frac{(t-s)^{\frac{3}{2}}}{\Gamma(\frac{5}{2})} \left(\frac{\exp(-s^2)}{32 \sqrt{1+s^2}} \frac{\sin u(s)+\sin v(s)}{16(s \exp(s^2)+1)} \right) ds, t \in [0, 1], \\ u(0) = \sqrt{3}, v(0) = \sqrt{2}, \end{cases} \quad (4.1)$$

We have $\alpha = \beta = \frac{1}{2}, \alpha_1 = \frac{3}{2}, \beta_1 = \frac{5}{2}, a = \sqrt{3}, b = \sqrt{2}, f_1(t, u, v) = \frac{\sin(u+v)}{18(\ln(t+1)+1)} + 6, f_2(t, u, v) = \frac{\sin u + \sin v}{16(t \exp(t^2)+6)}, \varphi_1(t) = \frac{\exp(-t)}{18(t+1)}$ and $\phi_1(t) = \frac{\exp(-t^2)}{32\sqrt{1+t^2}}$. Also, for $(u_1, v_1), (u_2, v_2) \in \mathbb{R}^2, t \in [0, 1]$, we have

$$|f_1(t, u_2, v_2) - f_1(t, u_1, v_1)| \leq \frac{1}{18} (|u_2 - u_1| + |v_2 - v_1|),$$

$$|f_2(t, u_2, v_2) - f_2(t, u_1, v_1)| \leq \frac{1}{16} (|u_2 - u_1| + |v_2 - v_1|),$$

$$|g_1(t, u_2, v_2) - g_1(t, u_1, v_1)| \leq \frac{1}{18} (|u_2 - u_1| + |v_2 - v_1|),$$

$$|h_1(t, u_2, v_2) - h_1(t, u_1, v_1)| \leq \frac{1}{16} (|u_2 - u_1| + |v_2 - v_1|).$$

Hence, $M_1 = 2.271, M_2 = 2.261, \mu_1 = \nu_1 = \frac{1}{18}, \mu_2 = \nu_2 = \frac{1}{16}, l_1 = m_1 = \frac{1}{18}, n_1 = k_1 = \frac{1}{16}$. Thus, we obtain $\bar{L} = \frac{1}{16}, \bar{L}(M_1 + M_2) = 0.283$. The conditions of the Theorem 3.1 hold. Therefore, the problem (4.1) has a unique solution on $[0, 1]$.

Example 4.2. Consider the following problem:

$$\begin{cases} D^{\frac{3}{4}} u(t) = e^t \cos(u(t)v(t)) + \ln(t+4) \\ \quad + \int_0^t \frac{(t-s)^{\sqrt{11}-1}}{\Gamma(\sqrt{11})} \left[\frac{s}{e^s} \cos(su(s)v(s)) \right] ds, t \in [0, 1], \\ D^{\frac{5}{7}} v(t) = \sinh(-\pi t^2 |u(t)v(t)|) \\ \quad + \int_0^t \frac{(t-s)^{\sqrt{7}-1}}{\Gamma(\sqrt{7})} \left[\sqrt{s} \exp(-|u(s)| - |v(s)|) \right] ds, t \in [0, 1], \\ u(0) = \sqrt{2}, v(0) = \sqrt{5}. \end{cases} \quad (4.2)$$

For this example, we have $\alpha = \frac{3}{4}, \beta = \frac{5}{7}, a = \sqrt{2}, b = \sqrt{5}$, and for all $t \in [0, 1], \varphi_1(t) = \frac{t}{e^t}, \phi_1(t) = \sqrt{t}$, and for each $(u, v) \in \mathbb{R}^2$,

$$\begin{aligned} f_1(t, u, v) &= e^t \cos(uv) + \ln(t+4), \\ f_2(t, u, v) &= \sinh(-\pi t^2 |uv|). \end{aligned}$$

The conditions of Theorem 3.2 hold. Then (4.2) has at least one solution on $[0, 1]$.

References

- Abdellaoui, M. A., Z. Dahmani and M. Houas (2013). On some boundary value problems for coupled system of arbitrary order. *Indian Journal of industrial and applied mathematics* **4**(2), 180–188.
- Anber, A., S. Belarbi and Z. Dahmani (2013). New existence and uniqueness results for fractional differential equations. *An. St. Univ. Ovidius Constanta* **21**(3), 33–41.
- Bai, C. Z. and J. X. Fang (2004). The existence of a positive solution for a singular coupled system of nonlinear fractional differential equations. *Applied Mathematics and Computation* **150**(3), 611–621.
- Bengrine, M. E. and Z. Dahmani (2012). Boundary value problems for fractional differential equations. *Int. J. Open problems compt. Math* **5**(4), 7–15.

- Cui, Z., P. Yu and Z. Mao (2012). Existence of solutions for nonlocal boundary value problems of nonlinear fractional differential equations. *Advances in Dynamical Systems and Applications* **7**(1), 31–40.
- Gaber, M. and M. G. Brikaa (2012). Existence results for a couple system of nonlinear fractional differential equation with three point boundary conditions. *Journal of Fractional Calculus and Applications* **3**(21), 1–10.
- Gafiychuk, V., B. Datsko and V. Meleshko (2008). Mathematical modeling of time fractional reaction-diffusion systems. *Journal of Computational and Applied Mathematics* **220**(1-2), 215–225.
- Kilbas, A. A. and S. A. Marzan (2005). Nonlinear differential equation with the Caputo fractional derivative in the space of continuously differentiable functions. *Differ. Equ* **41**(1), 84–89.
- Lakshmikantham, V. and A. S. Vatsala (2008). Basic theory of fractional differential equations. *Nonlinear Anal* **69**(8), 2677–2682.
- Mainardi, F. (1997). Fractional calculus: Some basic problem in continuum and statistical mechanics. Fractals and fractional calculus in continuum mechanics. *Springer, Vienna*.
- Su, X. (2009). Boundary value problem for a coupled system of nonlinear fractional differential equations. *Applied Mathematics Letters* **22**(1), 64–69.
- Wang, J., H. Xiang and Z. Liu (2010). Positive solution to nonzero boundary values problem for a coupled system of nonlinear fractional differential equations. *International Journal of Differential Equations* **2010**, 1–12.
- Zhang, S. (2006). Positive solution for boundary value problem of nonlinear fractional differential equations. *Electron. J. Differential Equations* **2006**(36), 1–12.



On BV_σ I-convergent Sequence Spaces Defined by an Orlicz Function

Vakeel. A. Khan^{a,*}, Mohd Shafiq^a, Rami Kamel Ahmad Rababah^a

^aDepartment of Mathematics A.M.U, Aligarh-202002, India

Abstract

In this article we study ${}_0BV_\sigma^I(M)$, $BV_\sigma^I(M)$ and ${}_\infty BV_\sigma^I(M)$ sequence spaces with the help of BV_σ space see (Mursaleen, 1983b) and an Orlicz function M . we study some topological and algebraic properties of these spaces and prove some inclusion relations.

Keywords: bounded variation, invariant mean, σ -Bounded variation, ideal, filter, Orlicz function, I-convergence, I-null, solid space, sequence algebra, convergence free space.

2010 MSC: 41A10, 41A25, 41A36, 40A30.

1. Introduction and Preliminaries

Let \mathbb{N} , \mathbb{R} and \mathbb{C} be the sets of all natural, real and complex numbers respectively. We denote

$$\omega = \{x = (x_k) : x_k \in \mathbb{R} \text{ or } \mathbb{C}\}$$

the space of all real or complex sequences. Let ℓ_∞ , c and c_0 denote the Banach spaces of bounded, convergent and null sequences respectively with norm $\|x\| = \sup_k |x_k|$.

Let v be denote the space of sequences of bounded variation. That is,

$$v = \left\{ x = (x_k) : \sum_{k=0}^{\infty} |x_k - x_{k-1}| < \infty, x_{-1} = 0 \right\}, \quad (1.1)$$

v is a Banach Space normed by $\|x\| = \sum_{k=0}^{\infty} |x_k - x_{k-1}|$ (Mursaleen, 1983b). Let σ be an injective mapping of the set of the positive integers into itself having no finite orbits. A continuous linear functional ϕ on ℓ_∞ is said to be an invariant mean or σ -mean if and only if:

*Corresponding author

Email addresses: vakhanmaths@gmail.com (Vakeel. A. Khan), shafiqmaths7@gmail.com (Mohd Shafiq), rami215r@hotmail.com (Rami Kamel Ahmad Rababah)

1. $\phi(x) \geq 0$ where the sequence $x = (x_k)$ has $x_k \geq 0$ for all k ,
2. $\phi(e) = 1$ where $e = \{1, 1, 1, \dots\}$,
3. $\phi(x_{\sigma(n)}) = \phi(x)$ for all $x \in \ell_\infty$.

If $x = (x_k)$, write $Tx = (Tx_k) = (x_{\sigma(k)})$. It can be shown that

$$V_\sigma = \left\{ x = (x_k) : \lim_{m \rightarrow \infty} t_{m,k}(x) = L \text{ uniformly in } k, L = \sigma - \lim x \right\} \quad (1.2)$$

where $m \geq 0, k > 0$.

$$t_{m,k}(x) = \frac{x_k + x_{\sigma(k)} \dots + x_{\sigma^m(k)}}{m+1} \text{ and } t_{-1,k} = 0, \quad (1.3)$$

where $\sigma_m(k)$ denote the m^{th} -iterate of $\sigma(k)$ at k . In case σ is the translation mapping, that is, $\sigma(k) = k+1$, σ -mean is called a Banach limit (Banach, 1932) and V_σ , the set of bounded sequences of all whose invariant means are equal, is the set of almost convergent sequences. The special case of (1.2) in which $\sigma(k) = k+1$ was given by (Lorentz, 1948), (Theorem 1), and that the general result can be proved in a similar way. It is familiar that a Banach limit extends the limit functional on c in the sense that

$$\phi(x) = \lim x, \text{ for all } x \in c. \quad (1.4)$$

Remark. In view of above discussion we have $c \subset V_\sigma$.

Theorem 1.1. A σ -mean extends the limit functional on c in the sense that $\phi(x) = \lim x$ for all $x \in c$ if and only if σ has no finite orbits. That is, if and only if for all $k \geq 0, j \geq 1, \sigma^j(k) \neq k$.

Put

$$\phi_{m,k}(x) = t_{m,k}(x) - t_{m-1,k}(x), \quad (1.5)$$

assuming that $t_{-1,k}(x) = 0$.

A straight forward calculation shows that (Mursaleen, 1983a)

$$\phi_{m,k}(x) = \begin{cases} \frac{1}{m(m+1)} \sum_{j=1}^m j(x_{\sigma^j(k)} - x_{\sigma^{j-1}(k)}), & \text{if } (m \geq 1), \\ x_k & \text{if } (m = 0) \end{cases}. \quad (1.6)$$

For any sequence x, y and scalar λ , we have $\phi_{m,k}(x+y) = \phi_{m,k}(x) + \phi_{m,k}(y)$ and $\phi_{m,k}(\lambda x) = \lambda \phi_{m,k}(x)$.

Definition 1.1. A sequence $x \in \ell_\infty$ is of σ -bounded variation if and only if

- (i) $\sum_{m=0}^{\infty} |\phi_{m,k}(x)|$ converges uniformly in k .
- (ii) $\lim_{m \rightarrow \infty} t_{m,k}(x)$, which must exist, should take the same value for all k .

Subsequently invariant means have been studied by (Mursaleen, 1983b,a; Ahmad & Mursaleen, 1986; Raimi, 1963; Khan & Ebadullah, 2013, 2012; Schafer, 1972) and many others. (Mursaleen, 1983b) defined the sequence space BV_σ , the space of all sequences of σ -bounded variation as $BV_\sigma = \{x \in \ell_\infty : \sum_m |\phi_{m,k}(x)| < \infty, \text{ uniformly in } k\}$.

Theorem 1.2. (*Fast, 1951*) BV_σ is a Banach space normed by $\|x\| = \sup_k \sum |\phi_{m,k}(x)|$.

Definition 1.2. (see[23]) A function $M : [0, \infty) \rightarrow [0, \infty)$ is said to be an Orlicz function if it satisfies the following conditions;

- (i) M is continuous, convex and non-decreasing,
- (ii) $M(0) = 0$, $M(x) > 0$ and $M(x) \rightarrow \infty$ as $x \rightarrow \infty$.

Remark. (see (*Tripathy & Hazarika, 2011*)) If the convexity of an Orlicz function is replaced by $M(x+y) \leq M(x) + M(y)$, then this function is called modulus function.

Remark. If M is an Orlicz function, then $M(\lambda x) \leq \lambda M(x)$ for all λ with $0 < \lambda < 1$.

An Orlicz function M is said to satisfy Δ_2 – Condition for all values of u if there exists a constant $K > 0$ such that $M(Lu) \leq KLM(u)$ for all values of $L > 1$ (see (*Tripathy & Hazarika, 2011*)).

(*Lindenstrauss & Tzafriri, 1971*) used the idea of an Orlicz function to construct the sequence space $\ell_M = \{x \in \omega : \sum_{k=1}^{\infty} M(\frac{|x_k|}{\rho}) < \infty, \text{ for some } \rho > 0\}$. The space ℓ_M becomes a Banach space with the norm

$$\|x\| = \inf\{\rho > 0 : \sum_{k=1}^{\infty} M(\frac{|x_k|}{\rho}) \leq 1\}, \quad (1.7)$$

which is called an Orlicz sequence space. The space ℓ_M is closely related to the space ℓ_p which is an Orlicz sequence space with $M(t) = t^p$ for $1 \leq p < \infty$.

Later on, some Orlicz sequence spaces were investigated by (*Hazarika & Esi, 2013; Maddox, 1970; Parshar & Choudhary, 1994; Bhardwaj & Singh, 2000; Et, 2001; Tripathy & Hazarika, 2011*) and many others.

Initially, as a generalization of statistical convergence (*Fridy, 1985*), the notation of ideal convergence (I-convergence) was introduced and studied by (*P. Kostyrko & Wilczyński, 2000*). Later on, it was studied by (*Khan & Ebadullah, 2013*), (*Hazarika & Esi, 2013; T. Šalát & Ziman, 2004, 2005*) and many others.

Here we give some preliminaries about the notion of I-convergence.

Definition 1.3. A sequence $x = (x_k) \in \omega$ is said to be statistically convergent to a limit $L \in \mathbb{C}$ if for every $\epsilon > 0$, we have $\lim_k \frac{1}{k} |\{n \in \mathbb{N} : |x_k - L| \geq \epsilon, n \leq k\}| = 0$, where vertical lines denote the cardinality of the enclosed set.

Definition 1.4. Let \mathbb{N} be the set of natural numbers. Then a family of sets $I \subseteq 2^{\mathbb{N}}$ (power set of \mathbb{N}) is said to be an ideal if:

- 1) I is additive i.e $\forall A, B \in I \Rightarrow A \cup B \in I$,
- 2) I is hereditary i.e $\forall A \in I \text{ and } B \subseteq A \Rightarrow B \in I$.

Definition 1.5. A non-empty family of sets $\mathcal{F}(I) \subseteq 2^{\mathbb{N}}$ is said to be filter on \mathbb{N} if and only if

- 1) $\Phi \notin \mathcal{F}(I)$,
- 2) $\forall A, B \in \mathcal{F}(I)$ we have $A \cap B \in \mathcal{F}(I)$,
- 3) $\forall A \in \mathcal{F}(I)$ and $A \subseteq B \Rightarrow B \in \mathcal{F}(I)$.

Definition 1.6. An Ideal $I \subseteq 2^{\mathbb{N}}$ is called non-trivial if $I \neq 2^{\mathbb{N}}$.

Definition 1.7. A non-trivial ideal $I \subseteq 2^{\mathbb{N}}$ is called admissible if $\{\{x\} : x \in \mathbb{N}\} \subseteq I$.

Definition 1.8. A non-trivial ideal I is maximal if there cannot exist any non-trivial ideal $J \neq I$ containing I as a subset.

Definition 1.9. For each ideal I , there is a filter $\mathfrak{F}(I)$ corresponding to I .
i.e $\mathfrak{F}(I) = \{K \subseteq \mathbb{N} : K^c \in I\}$, where $K^c = \mathbb{N} \setminus K$.

Definition 1.10. A sequence $x = (x_k) \in \omega$ is said to be I -convergent to a number L if for every $\epsilon > 0$, the set $\{k \in \mathbb{N} : |x_k - L| \geq \epsilon\} \in I$.
In this case, we write $I - \lim x_k = L$.

Definition 1.11. A sequence $x = (x_k) \in \omega$ is said to be I -null if $L = 0$. In this case, we write $I - \lim x_k = 0$.

Definition 1.12. A sequence $x = (x_k) \in \omega$ is said to be I -cauchy if for every $\epsilon > 0$ there exists a number $m = m(\epsilon)$ such that $\{k \in \mathbb{N} : |x_k - x_m| \geq \epsilon\} \in I$.

Definition 1.13. A sequence space E is said to be solid(normal) if $(\alpha_k x_k) \in E$ whenever $(x_k) \in E$ and for any sequence (α_k) of scalars with $|\alpha_k| \leq 1$, for all $k \in \mathbb{N}$.

Definition 1.14. A sequence space E is said to be symmetric if $(x_{\pi(k)}) \in E$ whenever $x_k \in E$. where π is a permutation on \mathbb{N} .

Definition 1.15. A sequence space E is said to be sequence algebra if $(x_k) * (y_k) = (x_k \cdot y_k) \in E$ whenever $(x_k), (y_k) \in E$.

Definition 1.16. A sequence space E is said to be convergence free if $(y_k) \in E$ whenever $(x_k) \in E$ and $x_k = 0$ implies $y_k = 0$, for all k .

Definition 1.17. Let $K = \{k_1 < k_2 < k_3 < k_4 < k_5 \dots\} \subset \mathbb{N}$ and E be a Sequence space. A K -step space of E is a sequence space $\lambda_K^E = \{(x_{k_n}) \in \omega : (x_k) \in E\}$.

Definition 1.18. A canonical pre-image of a sequence $(x_{k_n}) \in \lambda_K^E$ is a sequence $(y_k) \in \omega$ defined by

$$y_k = \begin{cases} x_k, & \text{if } k \in K, \\ 0, & \text{otherwise.} \end{cases}$$

A canonical preimage of a step space λ_K^E is a set of preimages all elements in λ_K^E . i.e. y is in the canonical preimage of λ_K^E iff y is the canonical preimage of some $x \in \lambda_K^E$.

Definition 1.19. A sequence space E is said to be monotone if it contains the canonical preimages of its step space.

Remark. If $I = I_f$, the class of all finite subsets of \mathbb{N} . Then, I is an admissible ideal in \mathbb{N} and I_f convergence coincides with the usual convergence.

Definition 1.20. If $I = I_\delta = \{A \subseteq \mathbb{N} : \delta(A) = 0\}$. Then, I is an admissible ideal in \mathbb{N} and we call the I_δ -convergence as the logarithmic statistical convergence.

Definition 1.21. If $I = I_d = \{A \subseteq \mathbb{N} : d(A) = 0\}$. Then, I is an admissible ideal in \mathbb{N} and we call the I_d -convergence as the asymptotic statistical convergence.

Remark. If $I_\delta - \lim x_k = l$, then $I_d - \lim x_k = l$.

The following lemmas remained an important tool for the establishment of some results of this article.

Lemma 1.1. (Tripathy & Hazarika, 2011). Every solid space is monotone.

Lemma 1.2. Let $K \in \mathcal{I}(I)$ and $M \subseteq \mathbb{N}$. If $M \notin I$, then $M \cap K \notin I$.

Lemma 1.3. If $I \subseteq 2^\mathbb{N}$ and $M \subseteq N$. If $M \notin I$, then $M \cap N \notin I$.

2. Main Results

Recently (Khan & Ebadullah, 2012) introduced and studied the following sequence space. For $m \geq 0$

$$BV_\sigma^I = \left\{ x = (x_k) \in \omega : \{k \in \mathbb{N} : |\phi_{m,k}(x) - L| \geq \epsilon\} \in I, \text{ for some } L \in \mathbb{C} \right\}. \quad (2.1)$$

In this article we introduce the following sequence spaces. For $m \geq 0$

$$BV_\sigma^I(M) = \left\{ x = (x_k) \in \omega : I - \lim M\left(\frac{|\phi_{m,k}(x) - L|}{\rho}\right) = 0, \text{ for some } L \in \mathbb{C}, \rho > 0 \right\}, \quad (2.2)$$

$${}_0BV_\sigma^I(M) = \left\{ x = (x_k) \in \omega : I - \lim M\left(\frac{|\phi_{m,k}(x)|}{\rho}\right) = 0, \rho > 0 \right\}, \quad (2.3)$$

$${}_\infty BV_\sigma^I(M) = \left\{ x = (x_k) \in \omega : \{k \in \mathbb{N} : \exists K > 0 \text{ s.t. } M\left(\frac{|\phi_{m,k}(x)|}{\rho}\right) \geq K\} \in I, \rho > 0 \right\}, \quad (2.4)$$

$${}_\infty BV_\sigma(M) = \left\{ x = (x_k) \in \omega : \sup M\left(\frac{|\phi_{m,k}(x)|}{\rho}\right) < \infty, \rho > 0 \right\}. \quad (2.5)$$

We also denote $\mathcal{M}_{BV_\sigma}^I(M) = BV_\sigma^I(M) \cap {}_\infty BV_\sigma(M)$ and ${}_0\mathcal{M}_{BV_\sigma}^I(M) = {}_0BV_\sigma^I(M) \cap {}_\infty BV_\sigma(M)$.

Throughout the article, if required, we denote $\phi_{m,k}(x) = x'_k$, $\phi_{m,k}(y) = y'_k$ and $\phi_{m,k}(z) = z'_k$ where x, y, z are $(x_k), (y_k)$ and (z_k) respectively.

Theorem 2.1. For any Orlicz function M , the classes of sequence ${}_0BV_\sigma^I(M)$, $BV_\sigma^I(M)$, ${}_0\mathcal{M}_{BV_\sigma}^I(M)$ and $\mathcal{M}_{BV_\sigma}^I(M)$ are the linear spaces.

Proof. We shall prove the result for the space $BV_\sigma^I(M)$, others will follow similarly.

For, let $x = (x_k), y = (y_k) \in BV_\sigma^I(M)$ be any two arbitrary elements and let α, β are scalars.

Now, since $(x_k), (y_k) \in BV_\sigma^I(M) \Rightarrow \exists$ some positive numbers $L_1, L_2 \in \mathbb{C}$ and $\rho_1, \rho_2 > 0$ such that

$$I - \lim_k M\left(\frac{|\phi_{m,k}(x) - L_1|}{\rho_1}\right) = 0, \quad (2.6)$$

$$I - \lim_k M\left(\frac{|\phi_{m,k}(y) - L_2|}{\rho_2}\right) = 0 \quad (2.7)$$

\Rightarrow for any given $\epsilon > 0$, the sets

$$A_1 = \left\{k \in \mathbb{N} : M\left(\frac{|\phi_{m,k}(x) - L_1|}{\rho_1}\right) > \frac{\epsilon}{2}\right\} \in I, \quad (2.8)$$

$$A_2 = \left\{k \in \mathbb{N} : M\left(\frac{|\phi_{m,k}(y) - L_2|}{\rho_2}\right) > \frac{\epsilon}{2}\right\} \in I. \quad (2.9)$$

Let

$$\rho_3 = \max\{2|\alpha|\rho_1, 2|\beta|\rho_2\}. \quad (2.10)$$

Since, M is non-decreasing and convex function, we have

$$M\left(\frac{|\alpha x'_k + \beta y'_k - (\alpha L_1 + \beta L_2)|}{\rho_3}\right) \leq M\left(\frac{|\alpha| |x'_k - L_1|}{\rho_3}\right) + M\left(\frac{|\beta| |y'_k - L_2|}{\rho_3}\right) \leq M\left(\frac{|x'_k - L_1|}{\rho_1}\right) + M\left(\frac{|y'_k - L_2|}{\rho_2}\right). \quad (2.11)$$

Therefore, from (2.8), (2.9) and (2.11), we have $\left\{k \in \mathbb{N} : M\left(\frac{|(\alpha x'_k + \beta y'_k) - (\alpha L_1 + \beta L_2)|}{\rho_3}\right) > \epsilon\right\} \subseteq A_1 \cup A_2 \in I$

implies that $\left\{k \in \mathbb{N} : M\left(\frac{|(\alpha x'_k + \beta y'_k) - (\alpha L_1 + \beta L_2)|}{\rho_3}\right) > \epsilon\right\} \in I$. That is, $I - \lim M\left(\frac{|(\alpha x'_k + \beta y'_k) - (\alpha L_1 + \beta L_2)|}{\rho_3}\right) = 0$. Thus,

$\alpha x_k + \beta y_k \in BV_\sigma^I(M)$. But $(x_k), (y_k) \in BV_\sigma^I(M)$ are the arbitrary elements. Therefore, $\alpha x_k + \beta y_k \in BV_\sigma^I(M)$, for all $(x_k), (y_k) \in BV_\sigma^I(M)$ and for all scalars α, β . Hence, $BV_\sigma^I(M)$ is linear. \square

Theorem 2.2. Let M_1 and M_2 be two Orlicz functions and satisfying Δ_2 - Condition, then

(a) $\mathcal{X}(M_2) \subseteq \mathcal{X}(M_1 M_2)$,

(b) $\mathcal{X}(M_1) \cap \mathcal{X}(M_2) \subseteq \mathcal{X}(M_1 + M_2)$ for $\mathcal{X} = {}_0BV_\sigma^I, BV_\sigma^I, {}_0\mathcal{M}_{BV_\sigma}^I$ and $\mathcal{M}_{BV_\sigma}^I$.

Proof. (a) Let $x = (x_k) \in {}_0BV_\sigma^I(M_2)$ be any arbitrary element $\Rightarrow \exists \rho > 0$ such that

$$I - \lim M_2\left(\frac{|\phi_{m,k}(x)|}{\rho}\right) = 0, \quad (2.12)$$

i.e.

$$I - \lim M_2\left(\frac{|x'_k|}{\rho}\right) = 0. \quad (2.13)$$

Let $\epsilon > 0$ and choose δ with $0 < \delta < 1$ such that $M_1(t) < \epsilon$, $0 \leq t \leq \delta$. Let us write $y_k = M_2(\frac{|x'_k|}{\rho})$ and consider

$$\lim_k M_1(y_k) = \lim_{y_k \leq \delta, k \in \mathbb{N}} M_1(y_k) + \lim_{y_k > \delta, k \in \mathbb{N}} M_1(y_k). \quad (2.14)$$

Now, since M_1 is an Orlicz function, we have $M_1(\lambda x) \leq \lambda M_1(x)$ for all λ with $0 < \lambda < 1$. Therefore,

$$\lim_{y_k \leq \delta, k \in \mathbb{N}} M_1(y_k) \leq M_1(2) \lim_{y_k \leq \delta, k \in \mathbb{N}} (y_k). \quad (2.15)$$

For $y_k > \delta$, we have $y_k < \frac{y_k}{\delta} < 1 + \frac{y_k}{\delta}$. Now, since M_1 is non-decreasing and convex, it follows that

$$M_1(y_k) < M_1(1 + \frac{y_k}{\delta}) < \frac{1}{2}M_1(2) + \frac{1}{2}M_1(\frac{2y_k}{\delta}). \quad (2.16)$$

Again, since M_1 satisfies Δ_2 – Condition, we have $M_1(y_k) < \frac{1}{2}K(\frac{y_k}{\delta})M_1(2) + \frac{1}{2}K(\frac{y_k}{\delta})M_1(2)$. Thus, $M_1(y_k) < K(\frac{y_k}{\delta})M_1(2)$. Hence,

$$\lim_{y_k > \delta, k \in \mathbb{N}} M_1(y_k) \leq \max\{1, K\delta^{-1}M_1(2)\} \lim_{y_k > \delta, k \in \mathbb{N}} (y_k). \quad (2.17)$$

Therefore, from (2.12), (2.13) and (2.14), we have $I\text{-}\lim_k M_1(y_k) = 0$, i.e. $I\text{-}\lim_k M_1 M_2(\frac{|\phi_{m,k}(x)|}{\rho}) = 0$, implies that $(x_k) \in {}_0BV_\sigma^I(M_1 M_2)$. Thus, ${}_0BV_\sigma^I(M_2) \subseteq {}_0BV_\sigma^I(M_1 M_2)$. Hence, $\mathcal{X}(M_2) \subseteq \mathcal{X}(M_1 M_2)$ for $\mathcal{X} = {}_0BV_\sigma^I$. For $\mathcal{X} = BV_\sigma^I$, $\mathcal{X} = {}_0M_{BV_\sigma}^I$ and $\mathcal{X} = \mathcal{M}_{BV_\sigma}^I$ the inclusions can be established similarly.

(b). Let $x = (x_k) \in {}_0BV_\sigma^I(M_1) \cap {}_0BV_\sigma^I(M_2)$. Let $\epsilon > 0$ be given. Then there exists $\rho > 0$ such that the sets $I\text{-}\lim M_1\left(\frac{|\phi_{m,k}(x)|}{\rho}\right) = 0$ and $I\text{-}\lim M_2\left(\frac{|\phi_{m,k}(x)|}{\rho}\right) = 0$. Therefore, $I\text{-}\lim M_1 + M_2\left(\frac{|\phi_{m,k}(x)|}{\rho}\right) = I\text{-}\lim M_1\left(\frac{|\phi_{m,k}(x)|}{\rho}\right) + I\text{-}\lim M_2\left(\frac{|\phi_{m,k}(x)|}{\rho}\right) = 0$. Thus, $x = (x_k) \in {}_0BV_\sigma^I(M_1 + M_2)$. Hence, ${}_0BV_\sigma^I(M_1) \cap {}_0BV_\sigma^I(M_2) \subseteq {}_0BV_\sigma^I(M_1 + M_2)$. For $\mathcal{X} = BV_\sigma^I$, $\mathcal{X} = {}_0M_{BV_\sigma}^I$ and $\mathcal{X} = \mathcal{M}_{BV_\sigma}^I$ the inclusions are similar. \square

For $M_2(x) = (x)$ and $M_1(x) = M(x)$, $\forall x \in [0, \infty)$, we have the following corollary.

Corollary 2.1. $\mathcal{X} \subseteq \mathcal{X}(M)$ for $\mathcal{X} = {}_0BV_\sigma^I$, BV_σ^I , ${}_0M_{BV_\sigma}^I$ and $\mathcal{M}_{BV_\sigma}^I$.

Theorem 2.3. For any orlicz function M , the spaces ${}_0BV_\sigma^I(M)$ and ${}_0M_{BV_\sigma}^I$ are solid and monotone.

Proof. Here we consider ${}_0BV_\sigma^I(M)$ and for ${}_0M_{BV_\sigma}^I$ the proof shall be similar. For, let $(x_k) \in {}_0BV_\sigma^I(M)$ be any arbitrary element. $\Rightarrow \exists \rho > 0$ such that $I\text{-}\lim_k M(\frac{|\phi_{m,k}(x)|}{\rho}) = 0$. Let (α_k) be a sequence of scalars such that $|\alpha_k| \leq 1$, for all $k \in \mathbb{N}$.

Now, since M is an Orlicz function. Therefore,

$$\begin{aligned} M\left(\frac{|\alpha_k \phi_{m,k}(x)|}{\rho}\right) &\leq |\alpha_k| M\left(\frac{|\phi_{m,k}(x)|}{\rho}\right) \leq M\left(\frac{|\phi_{m,k}(x)|}{\rho}\right) \\ \Rightarrow M\left(\frac{|\alpha_k \phi_{m,k}(x)|}{\rho}\right) &\leq M\left(\frac{|\phi_{m,k}(x)|}{\rho}\right), \text{ for all } k \in \mathbb{N}, \end{aligned}$$

implies that $I - \lim_k M\left(\frac{|\alpha_k \phi_{m,k}(x)|}{\rho}\right) = 0$.

Thus, $(\alpha_k x_k) \in {}_0BV_\sigma^I(M)$. Hence ${}_0BV_\sigma^I(M)$ is solid. Therefore, by lemma 1.1, ${}_0BV_\sigma^I(M)$ is monotone. Hence the result. \square

Theorem 2.4. For any orlicz function M , the spaces $BV_\sigma^I(M)$ and $\mathcal{M}_{BV_\sigma}^I$ are neither solid nor monotone in general.

Proof. Here we give counter example for the establishment of this result. For, let us consider $I = I_f$ and $M(x) = x$, for all $x \in [0, \infty)$. Consider, the K -step space $B_K(M)$ of $B(M)$ as follows. Let $(x_k) \in B(M)$ and $(y_k) \in B_K(M)$ be such that

$$y_k = \begin{cases} x_k, & \text{if } k \text{ is even,} \\ 0, & \text{otherwise.} \end{cases}$$

Consider the sequence (x_k) defined as $x_k = 1$, for all $k \in \mathbb{N}$, then $x_k \in BV_\sigma^I(M)$ and $\mathcal{M}_{BV_\sigma}^I$ but its K -step space pre-image does not belong to $BV_\sigma^I(M)$ and $\mathcal{M}_{BV_\sigma}^I$. Thus, $BV_\sigma^I(M)$ and $\mathcal{M}_{BV_\sigma}^I$ are not monotone and hence by lemma(I) they are not solid. \square

Theorem 2.5. For an Orlicz function M , the spaces ${}_0BV_\sigma^I(M)$ and $BV_\sigma^I(M)$ are not convergence free.

Proof. Let $I = I_f$ and $M(x) = x$ for all $x \in [0, \infty)$. Consider the sequences (x_k) and (y_k) defined as follows.

$$x_k = \frac{1}{k} \text{ and } y_k = k, \text{ for all } k \in \mathbb{N}.$$

Then, (x_k) belongs to both ${}_0BV_\sigma^I(M)$ and $BV_\sigma^I(M)$ but (y_k) does not belongs to both ${}_0BV_\sigma^I(M)$ and $BV_\sigma^I(M)$.

Hence, the spaces ${}_0BV_\sigma^I(M)$ and $BV_\sigma^I(M)$ are not convergence free. \square

Theorem 2.6. For an Orlicz function M , the spaces ${}_0BV_\sigma^I(M)$ and $BV_\sigma^I(M)$ are sequence algebra.

Proof. Here we consider ${}_0BV_\sigma^I(M)$. For the other one, result is similar.

Let $x = (x_k), y = (y_k) \in {}_0BV_\sigma^I(M)$ be any two arbitrary elements.

$\Rightarrow \exists \rho_1, \rho_2 > 0$ such that

$$I - \lim_k M\left(\frac{|\phi_{m,k}(x)|}{\rho_1}\right) = 0$$

and

$$I - \lim_k M\left(\frac{|\phi_{m,k}(y)|}{\rho_2}\right) = 0.$$

Let $\rho = \rho_1 \rho_2 > 0$. Then, it is obvious that $I - \lim_k M\left(\frac{|\phi_{m,k}(x)\phi_{m,k}(y)|}{\rho}\right) = 0$ implies that $(x_k \cdot y_k) = (x_k y_k) \in {}_0BV_\sigma^I(M)$. Hence, ${}_0BV_\sigma^I(M)$ is a Sequence algebra. \square

Theorem 2.7. Let M be an Orlicz function. Then, ${}_0BV_\sigma^I(M) \subseteq BV_\sigma^I(M) \subseteq {}_\infty BV_\sigma^I(M)$.

Proof. Let M be an Orlicz function. Then, we have to show that ${}_0BV_\sigma^I(M) \subseteq BV_\sigma^I(M) \subseteq {}_\infty BV_\sigma^I(M)$. Firstly, ${}_0BV_\sigma^I(M) \subseteq BV_\sigma^I(M)$ is obvious.

Now, let $x = (x_k) \in BV_\sigma^I(M)$ be any arbitrary element $\Rightarrow \exists \rho > 0$ such that $I\text{-}\lim_k M(\frac{|\phi_{m,k}(x)-L|}{\rho}) = 0$ for some $L \in \mathbb{N}$.

Now, $M(\frac{|\phi_{m,k}(x)|}{2\rho}) \leq \frac{1}{2}M(\frac{|\phi_{m,k}(x)-L|}{\rho}) + \frac{1}{2}M(\frac{|L|}{\rho})$. Taking supremum over k to both sides, we have $x = (x_k) \in {}_\infty BV_\sigma^I(M)$. Hence, ${}_0BV_\sigma^I(M) \subseteq BV_\sigma^I(M) \subseteq {}_\infty BV_\sigma^I(M)$. \square

Acknowledgments: The authors would like to record their gratitude to the reviewers for their careful reading and making some useful corrections which improved the presentation of the paper.

References

- Ahmad, Z. U and M. Mursaleen (1986). An application of Banach limits. *Proc. Amer. Math. Soc.*
- Banach, S. (1932). Théorie des opérations linéaires. *Warszawa*.
- Bhardwaj, V. K. and N. Singh (2000). Some sequence spaces defined by Orlicz functions. *Demonstratio Math.* **33**(3), 571–582.
- Et, M. (2001). On some new Orlicz spaces. *J. Analysis* **9**, 21–28.
- Fast, H. (1951). Sur la convergence statistique. **2**, 241–244.
- Fridy, J. A. (1985). On statistical convergence. *Analysis* **5**, 301–313.
- Hazarika, B. and A. Esi (2013). Some I-convergent generalized difference lacunary double sequence spaces defined by Orlicz function. *Acta Scientiarum. Technology* **35**(3), 527–537.
- Khan, V. A. and K. Ebadullah (2012). On a new I-convergent sequence space. *Analysis* **32**, 199–208.
- Khan, V. A. and K. Ebadullah (2013). On some new I-convergent sequence space. *Mathematics, Aeterna* **3**(2), 151–159.
- Lindenstrauss, J. and L. Tzafriri (1971). On Orlicz sequence spaces. *Israel J. Math.* **101**, 379–390.
- Lorentz, G. G. (1948). A contribution to the theory of divergent series. *Acta Math.* **80**(6), 167–190.
- Maddox, I. J. (1970). *Elements of functional analysis*. Cambridge University Press.
- Mursaleen, M. (1983a). Matrix transformation between some new sequence spaces. *Houston J. Math.*
- Mursaleen, M. (1983b). On some new invariant matrix methods of summability. *Quart. J. Math. Oxford* **9**, 77–86.
- P. Kostyrko, T. Šalát and W. Wilczyński (2000). I-convergence. *Raial Analysis Analysis Exchange* **26**(2), 669–686.
- Parshar, S. D. and B. Choudhary (1994). Sequence spaces defined by Orlicz function. *Indian J. Pure Appl. Math.* **25**, 419–428.
- Raimi, R. A. (1963). Invariant means and invariant matrix method summability. *Duke J. Math* **30**, 81–94.
- Schafer, P. (1972). Infinite matrices and invariant means. *Proc. Amer. soc.* **36**, 104–110.
- T. Šalát, B.C. Tripathy and M. Ziman (2004). On some properties of I-convergence. *Tatra Mt. Math. Publ.* **28**, 279–286.
- T. Šalát, B.C. Tripathy and M. Ziman (2005). On I-convergence field. *Ital. J. Pure Appl. Math.* **17**, 45–54.
- Tripathy, B.C. and B. Hazarika (2011). Some I-convergent sequence spaces defined by Orlicz function. *Acta Mathematicae Applicatae Sinica* **27**(1), 149–154.



Properties of Stabilizing Computations

Mark Burgin^a

^a*University of California, Los Angeles 405 Hilgard Ave. Los Angeles, CA 90095*

Abstract

Models play an important role in the development of computer science and information technology applications. Turing machine is one of the most popular model of computing devices and computations. This model, or more exactly, a family of models, provides means for exploration of capabilities of information technology. However, a Turing machine stops after giving a result. In contrast to this, computers, networks and their software, such as an operating system, very often work without stopping but give various results. There are different modes of such functioning and Turing machines do not provide adequate models for these processes. One of the closest to halting computation is stabilizing computation when the output has to stabilize in order to become the result of a computational process. Such stabilizing computations are modeled by inductive Turing machines. In comparison with Turing machines, inductive Turing machines represent the next step in the development of computer science providing better models for contemporary computers and computer networks. At the same time, inductive Turing machines reflect pivotal traits of stabilizing computational processes. In this paper, we study relations between different modes of inductive Turing machines functioning. In particular, it is demonstrated that acceptance by output stabilizing and acceptance by state stabilizing are linguistically equivalent.

Keywords: computation, stability, Turing machine, inductive Turing machine, acceptance, mode of computation, equivalence.

2010 MSC: 68Q05.

2012 CCS: theory of computation, models of computation.

1. Introduction

Computer science studies computations by means of theoretical models. One of the most popular theoretical models of computation is Turing machine. It is central in computer science and in many applications, especially, when it is necessary to prove impossibility of an algorithmic solution to a problem (Rogers, 1987). The pivotal feature of a Turing machine is the necessity to stop after giving a result because all subsequent machine operations become superfluous.

However, computers, networks and their software, such as an operating system, very often work without stopping but give various results. There are different modes of such functioning and Turing machines do not provide adequate models for these processes (Burgin, 2005a). Stabilizing computation is one of the closest to halting computation computational modes when the output has to stabilize in order to become the result of a computational process. Such stabilizing computations

are efficiently modeled by inductive Turing machines (Burgin & Debnath, 2004, 2005; Burgin, 2005a, 2006; Burgin & Gupta, 2012).

In comparison with Turing machines, inductive Turing machines represent the next step in the development of computer science providing better modeling tools for contemporary computers and computer networks (Burgin, 2005a). In particular, even simple inductive Turing machines and other inductive Turing machines of the first order can solve the Halting Problem for Turing machines, while inductive Turing machines of higher orders can generate and decide the whole arithmetical hierarchy as it is proved in (Burgin, 2003). Even more, unrestricted inductive Turing machines with a structured memory have the same computing power as Turing machines with oracles (Burgin, 2005a). In addition, inductive Turing machines allow decreasing time of computations (Burgin, 1999). Being more powerful, inductive Turing machines allow essential reduction of Kolmogorov (algorithmic) complexity of finite objects (Burgin, 2004), as well as algorithmic complexity of mathematical and computational problems (Burgin, 2010a). It is also important that in contrast to Turing machines, which can work only with words (Turing machines with one one-dimensional tape), with finite systems of words (Turing machines with several one-dimensional tapes) and with arrays (Turing machines with multidimensional tapes), inductive Turing machines can work not only with finite and infinite words, systems of words and multidimensional arrays but also with more sophisticated data structures, such as graphs, functions, hierarchical structures and chains of named sets or named data.

Inductive Turing machines have found applications in algorithmic information theory and complexity studies (Burgin, 2004, 2007, 2010a), software testing (Burgin & Debnath, 2009; Burgin *et al.*, 2009), high performance computing (Burgin, 1999), machine learning (Burgin & Klinger, 2004), software engineering (Burgin & Debnath, 2004, 2005), computer networks (Burgin, 2006; Burgin & Gupta, 2012) and evolutionary computations (Burgin & Eberbach, 2008, 2009b,a, 2010, 2012). For instance, inductive Turing machines can perform all types of machine learning - TxtEx-learning, TxtFin-learning, TxtBC-learning, and TxtEx*-learning, (Beros, 2013). While the traditional approach to machine learning models learning processes using functions, e.g., limit partial recursive functions (Gold, 1967), inductive Turing machines are automata, which can compute values of the modeling functions.

An important area of tentative application of inductive Turing machines and other super-recursive algorithms is software development and maintenance. As Călinescu, et al, (Calinescu *et al.*, 2013) write, modern software systems are often complex, inevitably distributed, and operate in heterogeneous and highly dynamic environments. Examples of such systems include those from the service-oriented, cloud computing, and pervasive computing domains. In these domains, continuous change is the norm and therefore the software must also change accordingly. In many cases, the software is required to self-react by adapting its behavior dynamically, in order to ensure required levels of service quality in changing environments. As a result, conventional recursive algorithms, such as Turing machines, cannot provide efficient means for modeling software functioning and behavior. This can be achieved only by utilization of inductive Turing machines and other super-recursive algorithms. Thus, better knowledge of inductive Turing machines properties and regularities of their behavior allows their better utilization and application of these models of computation and computer systems.

The goal of this paper is to study inductive Turing machines as models of real computing

devices, functioning of which often results in stabilizing computations. Note that stability is an important property of a computing device, as well as of computational processes. By definition, inductive Turing machines give results if and only if their computational process stabilizes (Burgin, 2005a).

Each real computing device, e.g., a computer, has three components: an input device (devices), an output device (devices) and a processor or a multiprocessor, which consists of several processors. Consequently, there are three modes of component functioning: the input mode, output mode and processing mode. Together they form the functioning mode of the computing device. For instance, the processing mode of an automaton can be acceptance (of the input), decision (about the input) or computation (of the final result). The output mode of a pushdown automaton can be acceptance by final state or acceptance by empty stack (Hopcroft *et al.*, 2001). The output mode of a Turing machine can be acceptance by final state or acceptance by halting (Hopcroft *et al.*, 2001). An inductive Turing machine can process information in a recursive mode or in the inductive mode (Burgin, 2005a). Computer scientists are usually interested in equivalence of different computational modes as it allows them to use the most appropriate mode for solving a given problem without loss of generality.

There are different types of equivalence: linguistic equivalence, functional equivalence, process equivalence, etc. (cf. (Burgin, 2010b)). Here we study the classical case of equivalence called linguistic equivalence.

We remind that two automata (computing devices) are linguistically equivalent if they have the same language (Burgin, 2010b). Note that it may be linguistic equivalence with respect to computation when the language of the automaton A is the language computed by A or it may be linguistic equivalence with respect to acceptance when the language of the automaton A is the language accepted by A.

We remind that two classes of automata are linguistically equivalent if they have the same classes of languages and two modes of functioning are linguistically equivalent if the classes of automata working in these modes are linguistically equivalent (Burgin, 2010b). As separate automata, classes of automata may be linguistically equivalent with respect to computation or with respect to acceptance. One of the basic results of the theory of pushdown automata is the statement that acceptance by final state is linguistically equivalent to acceptance by empty stack ((Hopcroft *et al.*, 2001) Section 6.2).

One of the basic results of the theory of Turing machines and recursive computations is the statement that acceptance by final state is linguistically equivalent to acceptance by halting ((Burgin, 2005a), Chapter 2).

The goal of this paper is to analyze different modes of inductive Turing machine functioning, finding whether similar results are true for these modes. Note that inductive Turing machines have much more modes of functioning than Turing machines. In Section 2, we remind some basic concepts and constructions from the theory of inductive Turing machines and stabilizing computations. In Section 3, we demonstrate that some key modes of inductive Turing machine functioning are linguistically equivalent.

2. Inductive Turing machines as models of stabilizing computations

To understand how inductive Turing machines model stabilizing computations, we need to know the hardware structure and characteristics of inductive Turing machine functioning in general and of the simple inductive Turing machine functioning, in particular, making the emphasis on work with finite words in some alphabet (Burgin, 2005a).

An inductive Turing machine M hardware consists of three abstract devices: a *control device* A , which is a finite automaton and controls performance of M ; a *processor or operating device* H , which corresponds to one or several *heads* of a conventional Turing machine; and the *memory* E , which corresponds to the *tape* or tapes of a conventional Turing machine. The memory E of the simplest inductive Turing machine consists of three linear tapes, and the operating device consists of three heads, each of which is the same as the head of a Turing machine and works with the corresponding tapes. Such machines are called *simple inductive Turing machines* (Burgin, 2005a).

The *control device* A is a finite automaton. It controls and regulates processes and parameters of the machine M : the state of the whole machine M , the processing of information by H , and the storage of information in the memory E .

The *memory* E of a general inductive Turing machines is divided into different but, as a rule, uniform cells. It is structured by a system of relations that organize memory as well-structured system and provide connections or ties between cells. In particular, *input* registers, the *working* memory, and *output* registers of M are discerned. Connections between cells form an additional structure K of E . Each cell can contain a symbol from an alphabet of the languages of the machine M or it can be empty.

In a general case, cells may be of different types. Different types of cells may be used for storing different kinds of data. For example, binary cells, which have type B, store bits of information represented by symbols 1 and 0. Byte cells (type BT) store information represented by strings of eight binary digits. Symbol cells (type SB) store symbols of the alphabet(s) of the machine M . Cells in conventional Turing machines have SB type. Natural number cells, which have type NN, are used in random access machines. Cells in the memory of quantum computers (type QB) store q-bits or quantum bits. Cells of the tape(s) of real-number Turing machines (Burgin, 2005a) have type RN and store real numbers. When different kinds of devices are combined into one, this new complex device may have several types of memory cells. In addition, different types of cells facilitate modeling the brain neuron structure by inductive Turing machines.

The *processor* H performs information processing in M . However, in comparison to computers, H performs very simple operations. When H consists of one unit, it can change a symbol in the cell that is observed by H , and go from this cell to another using a connection from K . It is possible that the processor H consists of several processing units similar to heads of a multihead Turing machine. This allows one to model various real and abstract computing systems: multiprocessor computers; Turing machines with several tapes; networks, grids and clusters of computers; cellular automata; neural networks; and systolic arrays.

The *software* R of the inductive Turing machine M is also a program that consists of simple rules:

$$q_h a_i \rightarrow a_j q_k c \quad (2.1)$$

Here q_h and q_k are states of A , a_i and a_j are symbols of the alphabet of M , and c is a type of connection in the memory E . The rule (2.1) means that if the state of the control device A of M is q_h and the processor H observes in the cell the symbol a_j , then the state of A becomes q_k , while the processor H writes the symbol a_j in the cell where it is situated and moves to the next cell by a connection of the type c . Each rule directs one step of computation of the inductive Turing machine M . Rules of the inductive Turing machine M define the transition function of M and describe changes of A , H , and E . Consequently, these rules also determine the transition functions of A , H , and E .

These rules cover only synchronous parallelism of computation. However, it is possible to consider inductive Turing machines of any order in which their processor can perform computations in the concurrent mode. Besides, it is also possible to consider inductive Turing machines with several processors. These models of computation are studied elsewhere.

A general step of the machine M has the following form. At the beginning, the processor H observes some cell with a symbol a_i (it may be Λ as the symbol of an empty cell) and the control device A is in some state q_h . Then the control device A (and/or the processor H) chooses from the system R of rules a rule r with the left part equal to $q_h a_i$ and performs the operation prescribed by this rule. If there is no rule in R with such a left part, the machine M stops functioning. If there are several rules with the same left part, M works as a nondeterministic Turing machine, performing all possible operations. When A comes to one of the final states from F , the machine M also stops functioning. In all other cases, it continues operation without stopping.

In the output stabilizing mode, M gives the result when M halts and its control device A is in a final state from F , or when M never stops but at some step of the computation the content of the output register becomes fixed and does not change (cf. Definition 3.4). The computed result of M is the word that is written in the output register of M . In all other cases, M does not give the result (cf. Definition 3.6).

Now let us build a constructive hierarchy of inductive Turing machines.

The memory E is called *recursive* if all relations that define its structure are recursive. Here recursive means that there are Turing machines that decide or build the structured memory (Burgin, 2005a). There are different techniques to organize this process. The simplest approach assumes that given some data, e.g., a description of the structure of E , a Turing machine T builds all connections in the memory E before the machine M starts its computation. According to another methodology, memory construction by the machine T and computations of the machine M go concurrently, while the machine M computes, the machine T constructs connections in the memory E . It is also possible to consider a situation when some connections in the memory E are assembled before the machine M starts its computation, while other connections are formed parallel to the computing process of the machine M .

Besides, it is possible to consider a schema when the machine T is separate from the machine M , while another construction adopts the machine T as a part of the machine M .

Inductive Turing machines with recursive memory are called *inductive Turing machines of the first order*.

While in inductive Turing machines of the first order, the memory is constructed by Turing machines or other recursive algorithms, it is possible to use inductive Turing machines for memory construction for other inductive Turing machines. This brings us to the concept of inductive Turing

machines of higher orders. For instance, in inductive Turing machines of the second order, the memory is constructed by Turing machines of the first order.

In general, we have the following definitions.

The memory E is called n -inductive if its structure is constructed by an inductive Turing machine of the order n . Inductive Turing machines with n -inductive memory are called *inductive Turing machines of the order $n + 1$* . Namely, in inductive Turing machines of order n , the memory is constructed by Turing machines of order $n - 1$.

We denote the class of all inductive Turing machines of the order n by \mathbf{IT}_n and take $\mathbf{IT} = \bigcup_{n=1}^{\infty} \mathbf{IT}_n$.

In such a way, we build a constructive hierarchy of inductive Turing machines. Algorithmic problems solved by these machines form a superrecursive hierarchy of algorithmic problems (Burgin, 2005b).

A simple inductive Turing machine has the same structure and the same rules (instructions) as a conventional Turing machine with three heads and three linear tapes: the input tape, output tape and working tape. The input tape is a read-only tape and the output tape is a write-only tape.

Computation or acceptation of a simple inductive Turing machine M consists of two stages. At first, M rewrites the input word from the input tape to the working tape. Then M starts working with this word in the working tape, writing something to the output tape from time to time.

Thus, the rules of a simple inductive Turing machine have the form

$$q_h(a_{i1}, a_{i2}, a_{i3}) \rightarrow (a_{j1}, a_{j2}, a_{j3})q_k(T_1, T_2, T_3) \quad (2.2)$$

The meaning of symbols in formula (2.2) is similar to notations used for Turing machines. Namely, we have:

- q_h and q_k are states of the control device A ;
- $a_{i1}, a_{i2}, a_{i3}, a_{j1}, a_{j2}$ and a_{j3} are symbols from the alphabet of M ;
- each of the symbols T_1, T_2 and T_3 is equal either to L , which denotes the transition of the head to the left adjacent cell or to R , which denotes the transition of the head to the right adjacent cell, or to N , which denotes absence of a head transition.

The rule (2.2) means that if the state of the control device A of M is q_h and the head h_t observes in the cell of the tape t the symbol a_{it} , then the state of A becomes q_k , while the head h_t writes the symbol a_j in the cell where it is situated and moves to the direction indicated by T_t ($t = 1, 2, 3$). Each rule directs one step of computation of the inductive Turing machine M .

It means that moves of a simple inductive Turing machine are the same as moves of a Turing machine with three tapes. The difference is in output. A Turing machine produces a result only when it halts. The result is a word on the output tape. A simple inductive Turing machine is also doing this but in addition, it produces its results without stopping. It is possible that in the sequence of computations after some step, the word on the output tape is not changing, while the simple inductive Turing machine continues working. This word, which is not changing, is the result of the machine that works in the output stabilizing computing mode. Thus, the simple

inductive Turing machine does not halt, producing a result after a finite number of computing operations.

Because here we consider inductive Turing machines that work only with finite words and processing of the input word starts only after it is rewritten into the working tape, it is possible to assume that there is no input tape, the initial word is written in the working tape and the rules have the form

$$q_h(a_{i2}, a_{i3}) \rightarrow (a_{j2}, a_{j3})q_k(T_2, T_3) \quad (2.3)$$

Here a_{i2} and a_{j2} are symbols in the working tape, a_{i3} and a_{j3} are symbols in the output tape, the symbol T_2 directs the move of the head h_2 , while the symbol T_3 directs the move of the head h_3 .

Besides, it is also possible to assume that the output written in the output tape does not influence operations of the inductive Turing machine because the output tape is the write-only tape.

3. Comparing results of stabilizing computations

In this section, we study functioning of inductive Turing machines of the first order with one linearly ordered output register and one linearly ordered input register (Burgin, 2005a). We also assume that these inductive Turing machines work with finite words in some alphabet. The main emphasis here is on simple inductive Turing machines. Note that in general, inductive Turing machines can work not only with finite and infinite words but also with multidimensional arrays, graphs and even more sophisticated data structures.

In the previous section, we considered only the output stabilizing computing mode of inductive Turing machines. However, it is possible to use other modes of inductive Turing machine functioning, for example, the state stabilizing mode. This is similar to the functioning modes of finite automata that work with infinite words, (Burks & Wright, 1953; Büchi, 1960; Thomas, 1990; Chadha *et al.*, 2009).

The simplest modes of inductive Turing machine functioning are acceptance by halting and computation by halting. Namely, we have:

Definition 3.1. a) An inductive Turing machine M accepts the input word *by halting* if after some number of steps, the machine M stops.

b) The set $L_{ht}(M)$ of all words accepted by halting of an inductive Turing machine M is called the *halting accepted language* of the machine M .

This is the standard mode for Turing machines (Hopcroft *et al.*, 2001; Burgin, 2005a).

Computation by halting is defined in a similar way.

Definition 3.2. a) An inductive Turing machine M computes a word w *by halting* if after some number of steps, the machine M stops and when this happens, w is the word in output tape.

b) The set $L^{ht}(M)$ of all words computed by halting of an inductive Turing machine M is called the *halting computed language* of the machine M .

In the theory of inductive Turing machines, it is proved that when an inductive Turing machine M gives the result by halting, i.e., accepts a word by halting or computes a word by halting, M is linguistically equivalent to a Turing machine, i.e., the language produced (accepted or computed) by M can be produced by some Turing machine (Burgin, 2005a). In the theory of Turing machines, it is proved that acceptance by halting is linguistically equivalent to computation by halting. This gives us the following result.

Proposition 3.1. *Computation by halting is linguistically equivalent to acceptance by halting in the class of all inductive Turing machines of the first order.*

Thus, inductive Turing machine M of the first order can compute and accept all recursively enumerable languages and only these languages.

Now we will study more productive modes of inductive Turing machine functioning when machines are able to compute or accept languages that are not recursively enumerable.

Let us consider an inductive Turing machine M . In the set Q of states of the machine M , several subsets F_1, \dots, F_k are selected and called the *final groups of states* of the inductive Turing machine M .

Definition 3.3. An inductive Turing machine M gives a *result by state stabilizing* if after some number of steps, the state of the machine M always remains in the same final group of states.

Note that traditionally states of the control device of a Turing machine or of an inductive Turing machine are treated as states of the whole machine (Burgin, 2005a; Hopcroft et al., 2001; Sipser, 1996). We follow this tradition.

It is possible that one or several final groups consist of a single state. Then stabilization in such a group means that the state of the machine M always stops changing after some number of steps.

When the processing mode is acceptance, the result is acceptance of the input word. We formalize this situation by the following definition.

Definition 3.4. a) An inductive Turing machine M accepts the input word *by state stabilizing* if after some number of steps, the state of the machine M always remains in the same final group of states.

b) The set $L_{st}(M)$ of all words accepted by state stabilizing of an inductive Turing machine M is called the *state stabilizing accepted language* of the machine M .

It is natural to consider the state stabilizing language $L_{st}(M)$ of the machine M as the result of M working in the acceptance mode.

We denote by $\mathbf{L}_{st}(\text{ITM1})$ the set of all state stabilizing accepted languages of inductive Turing machines of the first order and by $\mathbf{L}_{st}(\text{SITM})$ the set of all state stabilizing accepted languages of simple inductive Turing machines.

In the computing mode, the result is defined in a different way, which is formalized by the following definition.

- Definition 3.5.** a) An inductive Turing machine M computes the input word w by *state stabilizing* if after some number of steps, the state of the machine M always remains in the same final group of states and w is the first word in the output tape when the state stabilization process starts. This output w is the final result of the machine M .
- b) The set $L^{st}(M)$ of all words computed by state stabilizing of an inductive Turing machine M is called the *state stabilizing computed language* of the machine M .

It is natural to consider the state stabilizing language $L_{ot}(M)$ of the machine M as the result of M working in the acceptance mode.

We denote by $\mathbf{L}^{st}(\text{ITM1})$ the set of all state stabilizing computed languages of inductive Turing machines of the first order and by $\mathbf{L}^{st}(\text{SITM})$ the set of all state stabilizing computed languages of simple inductive Turing machines.

Definition 3.6. An inductive Turing machine M gives a *result by output stabilizing* if after some number of steps the output of the machine M stops changing. This output is the final result of the machine M .

When the processing mode is acceptance, the result is acceptance of the input word. Namely, we have the following concept.

- Definition 3.7.** a) An inductive Turing machine M accepts the input word by *output stabilizing* if after some number of steps, the output of the machine M stops changing. This output is the final result of the machine M .
- b) The set $L_{ot}(M)$ of all words accepted by output stabilizing of an inductive Turing machine M is called the *output stabilizing accepted language* of the machine M .

It is natural to consider the output stabilizing language $L_{ot}(M)$ of the machine M as the result of M working in the acceptance mode.

We denote by $\mathbf{L}_{ot}(\text{ITM1})$ the set of all output stabilizing accepted languages of inductive Turing machines of the first order and by $\mathbf{L}_{ot}(\text{SITM})$ the set of all state stabilizing accepted languages of simple inductive Turing machines.

- Definition 3.8.** a) An inductive Turing machine M computes the input word by *output stabilizing* if after some number of steps, the output of the machine M stops changing. This output is the final result of the machine M .
- b) The set $L^{ot}(M)$ of all words computed by output stabilizing of an inductive Turing machine M is called the *output stabilizing computed language* of the machine M .

It is natural to consider the state stabilizing language $L^{ot}(M)$ of the machine M as the result of M working in the computation mode.

We denote by $\mathbf{L}^{ot}(\text{ITM1})$ the set of all state stabilizing computed languages of inductive Turing machines of the first order and by $\mathbf{L}^{ot}(\text{SITM})$ the set of all state stabilizing computed languages of simple inductive Turing machines.

Let us consider more restrictive modes of inductive Turing machine functioning.

Definition 3.9. An inductive Turing machine M gives a *result by bistabilizing* if after some number of steps the output of the machine M stops changing, while the state of M remains in the same final group of states.

This output is the final result of the machine M .

When the processing mode is acceptance, the result is acceptance of the input word. Namely, we have the following concept.

Definition 3.10. a) An inductive Turing machine M accepts the input word *by bistabilizing* if after some number of steps, the output of the machine M stops changing, while the state of M remains in the same final group of states.

b) The set $L_{bt}(M)$ of all words accepted by bistabilizing of an inductive Turing machine M is called the *bistabilizing accepted language* of the machine M .

It is natural to consider the output stabilizing language $L_{bt}(M)$ of the machine M as the result of M working in the acceptance mode.

We denote by $\mathbf{L}_{bt}(\text{ITM1})$ the set of all bistabilizing accepted languages of inductive Turing machines of the first order and by $\mathbf{L}_{bt}(\text{SITM})$ the set of all bistabilizing accepted languages of simple inductive Turing machines.

Definition 3.11. a) An inductive Turing machine M computes the input word *by bistabilizing* if after some number of steps, the output of the machine M stops changing, while the state of M remains in the same final group of states.

The output that stopped changing is the final result of the machine M .

b) The set $L^{bt}(M)$ of all words computed by bistabilizing of an inductive Turing machine M is called the *bistabilizing computed language* of the machine M .

It is natural to consider the bistabilizing language $L^{bt}(M)$ of the machine M as the result of M working in the computation mode.

We denote by $\mathbf{L}^{bt}(\text{ITM1})$ the set of all bistabilizing computed languages of inductive Turing machines of the first order and by $\mathbf{L}^{bt}(\text{SITM})$ the set of all bistabilizing computed languages of simple inductive Turing machines.

Definitions imply the following results.

Proposition 3.2. For any inductive Turing machine M , we have:

a) $L^{bt}(M) \subseteq L^{ot}(M)$.

b) $L^{bt}(M) \subseteq L^{st}(M)$.

c) $L_{bt}(M) \subseteq L_{ot}(M)$.

d) $L_{bt}(M) \subseteq L_{st}(M)$.

Corollary 3.1. The following inclusions are true:

- a) $\mathbf{L}^{bt}(\text{ITM1}) \subseteq \mathbf{L}^{ot}(\text{ITM1})$.
- b) $\mathbf{L}_{bt}(\text{ITM1}) \subseteq \mathbf{L}_{ot}(\text{ITM1})$.
- c) $\mathbf{L}_{bt}(\text{ITM1}) \subseteq \mathbf{L}_{st}(\text{ITM1})$.
- d) $\mathbf{L}^{bt}(\text{ITM1}) \subseteq \mathbf{L}^{st}(\text{ITM1})$.
- e) $\mathbf{L}^{bt}(\text{SITM}) \subseteq \mathbf{L}^{ot}(\text{SITM})$.
- f) $\mathbf{L}_{bt}(\text{SITM}) \subseteq \mathbf{L}_{ot}(\text{SITM})$.
- g) $\mathbf{L}_{bt}(\text{SITM}) \subseteq \mathbf{L}_{st}(\text{SITM})$.
- h) $\mathbf{L}^{bt}(\text{SITM}) \subseteq \mathbf{L}^{st}(\text{SITM})$.

Computation by output stabilizing is the basic mode of inductive Turing machines when they perform computations (Burgin, 2005a). Properties of inductive Turing machines show that an inductive Turing machine M of the first order that accepts (or computes) by halting is linguistically equivalent to an accepting (or computing) Turing machine (Burgin, 2005a). At the same time, in general inductive Turing machines of the first order are essentially more powerful than Turing machines. For instance, there are simple inductive Turing machines that solve the halting problem for all Turing machines. This gives us the following result.

Proposition 3.3. *For any inductive Turing machine M , we have:*

- a) *Computation by state stabilizing is not linguistically equivalent to computation by halting in the class of all inductive Turing machines of the first order.*
- b) *Acceptation by state stabilizing is not linguistically equivalent to acceptance by halting in the class of all inductive Turing machines of the first order.*

Note that halting is a very specific case of stabilizing in which the process simply stops. However, it is more natural to compare non-stopping processes of acceptance and computation for inductive Turing machines.

Comparing the state stabilizing computed language $L^{st}(M)$ of an inductive Turing machine M and the output stabilizing computed language $L^{ot}(M)$ of the machine M , we see that in general these languages do not coincide. The same can be true for the accepted languages of the inductive Turing machine M . Such machines are considered in the following examples.

Example 3.1. Let us take a simple inductive Turing machine M such that works in the alphabet $\{0, 1\}$ and given a word w as its input, changes w to the word $w1$, i.e., M writes 1 at the end of w , gives as the output and repeats this operation with the output without stopping. As the output of M is changing on each step, the output stabilizing accepted language $L_{ot}(M)$ of the machine M is empty. At the same time, if we take all states of M as a single final group, then the state stabilizing accepted language $L_{st}(M)$ of the machine M contains all words in the alphabet $\{0, 1\}$. Thus, $L_{ot}(M) \neq L_{st}(M)$.

The same inequality is true for the computed language of the inductive Turing machine M . Indeed, the output stabilizing computed language $L^{ot}(M)$ of the machine M is empty because its output never stops changing. At the same time, the machine M computes by the state stabilizing all words in the alphabet $\{0, 1\}$ that has 1 at the end. Thus, $L^{ot}(M) \neq L^{st}(M)$.

In Example 3.1, as we can see, the state stabilizing accepted language $L_{st}(M)$ of the machine M is much larger than the output stabilizing accepted language $L_{ot}(M)$ of the same machine M . The same inequality is true for the computed languages of inductive Turing machine M . However, the opposite situation is also possible.

Example 3.2. Let us take a simple inductive Turing machine W such that works in the alphabet $\{0, 1\}$ and given a word w as its input, gives this word w as the output and repeats this operation with the output and then never produces any new output. At the same time, it is possible to program the machine W so that after it rewrites w into its output tape, the machine W goes into an infinite cycle changing the state on each step. As the result, the output stabilizing accepted language $L_{ot}(W)$ of the machine W contains all words in the alphabet $\{0, 1\}$. However, if we do not define any final group in the states of the machine W , then the state stabilizing accepted language $L_{st}(W)$ of the machine W is empty. Thus, $L_{ot}(W) \neq L_{st}(W)$.

The same inequality is true for the computed languages of inductive Turing machine W . Indeed, the state stabilizing computed language $L^{ot}(W)$ of the machine W is empty because its state never stops changing and there are no final state groups. At the same time, the machine W computes by the output stabilizing all words in the alphabet $\{0, 1\}$ that has 1 at the end. Thus, $L^{ot}(W) \neq L^{st}(W)$.

Thus, the output stabilizing accepted language $L_{ot}(W)$ of the machine W is much larger than the state stabilizing accepted language $L_{st}(W)$ of the same machine W . The same inequality is true for the computed languages of inductive Turing machine W .

However, for the classes of the output stabilizing accepted by inductive Turing machines of the first order languages and the state stabilizing accepted by inductive Turing machines of the first order languages this is not true.

Theorem 3.1. *If a language L has an inductive Turing machine of the first order that computes L by output stabilizing, then L has an inductive Turing machine of the first order that computes L by bistabilizing.*

Proof. Let us consider an inductive Turing machine M of the first order that computes a language L by output stabilizing and construct an inductive Turing machine of the first order that computes a language L by bistabilizing. In the theory of inductive Turing machines, it is proved that any an inductive Turing machine of the first order is equivalent to a simple inductive Turing machine M that never stops given some input (Burgin, 2005a). Thus, it is possible to assume that M is a simple inductive Turing machine that never stops given some input and accepts by output stabilizing. Note that in the mode of output stabilizing final groups of states do not play any role.

Thus, it is possible to use the same machine M for bistabilizing computation of L . Indeed, making the set Q of all states of M as one final group, we see that according to Definition 11, the machine M computes the same language $L(M)$ as before by bistabilizing because the state is always stabilized. Theorem is proved. \square

Results from (Burgin, 2005a) show that $\mathbf{L}_{st}(\text{ITM1}) = \mathbf{L}_{st}(\text{SITM})$ and $\mathbf{L}_{ot}(\text{ITM1}) = \mathbf{L}_{ot}(\text{SITM})$. Thus, Theorem 3.1 implies the following result.

Corollary 3.2. $\mathbf{L}^{ot}(\text{ITM1}) \subseteq \mathbf{L}^{bt}(\text{ITM1})$.

This result shows that computation by bistabilizing looks more powerful than computation by output stabilizing in the class of all inductive Turing machines of the first order. However, the inverse of Theorem 3.1 is also true.

Theorem 3.2. *If a language L has an inductive Turing machine M of the first order that computes L by bistabilizing, then L has an inductive Turing machine K of the first order that computes L by output stabilizing.*

Proof. Let us consider an inductive Turing machine M of the first order that computes a language L by bistabilizing and construct an inductive Turing machine K of the first order that computes a language L by output stabilizing. As before, it is possible to assume that M is a simple inductive Turing machine.

To allow functioning in the bistabilizing mode, in the set Q of states of M , several subsets F_1, \dots, F_k are selected as final groups of states of the inductive Turing machine M .

By Proposition 3.2, $L = L^{bt}(M) \subseteq L^{ot}(M)$, i.e., the language $L^{ot}(M)$ computable by output stabilizing may have words that do not belong to the language $L^{bt}(M)$ computable by bistabilizing. To prove the necessary result, we show how to get rid of these extra words by appropriately changing the machine M .

Such an extra word w is computed by M when the output stabilizes but the state does not remain in the same final group. To prevent this, we add new symbols $b_1, b_2, b_3, \dots, b_m$ to the alphabet $A = \{a_1, a_2, a_3, \dots, a_m\}$ of the machine M . Then we consider all instructions of the form

$$q_h(a_{i2}, a_{i3}) \rightarrow (a_{j2}, a_{i3})q_k(T_2, T_3)$$

where q_h belongs to some final group of states F_t , while q_k does not belong to this group and change this instruction to the two following instructions

$$q_h(a_{i2}, a_{i3}) \rightarrow (a_{j2}, b_{i3})q_k(T_2, T_3)$$

$$q_k(a_{i2}, b_{i3}) \rightarrow (a_{j2}, a_{i3})q_k(T_2, T_3)$$

We do not change other instructions of M and in such a way, we obtain a simple inductive Turing machine K . As a result of these changes, the output of K always changes when the state leaves some final group. So, the output stabilizes if and only if the state stabilizes in some final group. Consequently, the new inductive Turing machine K computes the given language L . Theorem is proved. \square

Theorem 3.2 implies the following result.

Corollary 3.3. $\mathbf{L}^{bt}(\text{ITM1}) \subseteq \mathbf{L}^{ot}(\text{ITM1})$.

This result shows that computation by output stabilizing looks more powerful than computation by bistabilizing in the class of all inductive Turing machines of the first order. However, Theorems 3.1 and 3.2 together imply that these modes are linguistically equivalent.

Theorem 3.3. *Computation by output stabilizing is linguistically equivalent to computation by bistabilizing in the class of all inductive Turing machines of the first order.*

Corollary 3.4. $L^{bt}(\text{ITM1}) = L^{ot}(\text{ITM1})$.

Let us consider relations between computed and accepted languages.

Theorem 3.4. *If a language L has an inductive Turing machine of the first order that computes L by state stabilizing, then L has an inductive Turing machine of the first order that computes L by bistabilizing.*

Proof. Let us consider an inductive Turing machine M of the first order that computes a language L by state stabilizing and construct an inductive Turing machine K of the first order that computes a language L by bistabilizing. As before, it is possible to assume that M is a simple inductive Turing machine.

To allow functioning in the state stabilizing mode, in the set Q of states of M , several subsets F_1, \dots, F_k are selected as final groups of states of the inductive Turing machine M .

We begin our construction of the machine K by adding one more working tape to the tapes the machine M . A simple inductive Turing machine has only three tapes (see Section 2). However, by the standard technique described, for example, in (Hopcroft *et al.*, 2001) or in (Sipser, 1996), it is possible to show that we can build a simple inductive Turing machine which simulates two working tapes using only one working tape and accepting the same language. Thus, adding one more working tape, we do not extend the class of accepted languages.

As a result of this action, the machine K has an input tape T_{in} , an output tape T_{out} and two working tapes T_{1w} and T_{2w} . We use the tape T_{1w} for exact modeling of the working tape T_w of the machine M . To do this, we preserve all parts of the rules that are related to the tape T_w in the machine M .

At the same time, we use the tape T_{2w} for exact modeling of the output tape T_{Mout} of the machine M . To do this, we redirect all parts of the rules that are related to the tape T_{Mout} in the machine M , making them the rules for the tape T_{2w} in K .

Besides, we add the rewriting state r to the set of states of the machine M and new rules for the tape T_{out} in K . The rules in which change of the state goes in one and the same final group are preserved in K . This allows the following operations. When the state of M leaves a final group, the same happens with the machine K according to its rules. While the machine M continues its functioning, comes to the rewriting state r , erases everything from the tape T_{out} in K and then rewrites the word from the tape T_{2w} into the output tape T_{out} . When the state of M is outside any final group and changes, the machine K repeats the same steps.

Thus, the output of K always changes when the state leaves some final group or is outside any final group and changes. As a result, the output of K stabilizes if and only if the state remains inside some final group. Besides, it stabilizes on the first word that was on the output tape of M when the stabilization of states started. So, the machine K computes the language L by bistabilizing. Theorem is proved. \square

Theorem 3.4 implies the following result.

Corollary 3.5. $L^{st}(\text{ITM1}) \subseteq L^{bt}(\text{ITM1})$.

Note that the machine K constructed in the proof of Theorem 3.4 also computes the language L by output stabilizing. This gives us the following result.

Theorem 3.5. *If a language L has an inductive Turing machine of the first order that computes L by state stabilizing, then L has an inductive Turing machine of the first order that computes L by output stabilizing.*

Theorem 3.5 implies the following result.

Corollary 3.6. $L^{st}(\text{ITM1}) \subseteq L^{ot}(\text{ITM1})$.

Inversion of Theorem 3.4 is also true.

Theorem 3.6. *If a language L has an inductive Turing machine of the first order that computes L by bistabilizing, then L has an inductive Turing machine of the first order that computes L by state stabilizing.*

Proof. Let us consider an inductive Turing machine M of the first order that computes a language L by bistabilizing and construct an inductive Turing machine K of the first order that computes a language L by state stabilizing. As before, it is possible to assume that M is a simple inductive Turing machine.

To allow functioning in the bistabilizing mode, in the set $Q = \{q_1, q_2, q_3, \dots, q_m\}$ of states of M , several subsets F_1, \dots, F_k are selected as final groups of states of the inductive Turing machine M .

By Proposition 3.2, $L = L^{bt}(M) \subseteq L^{st}(M)$, i.e., the language $L^{st}(M)$ computable by output stabilizing may have words that do not belong to the language $L^{bt}(M)$ computable by bistabilizing. To prove the necessary result, we show how to get rid of these extra words by appropriately changing the machine M .

Such an extra word w is computed by M when the state stabilizes in some final group but the output does not remain the same all the time. To prevent this, we add a new states $p_1, p_2, p_3, \dots, p_m$ to the set Q of states of the machine M without changing the final groups. Then we consider all instructions of the form.

$$q_h(a_{i2}, a_{i3}) \rightarrow (a_{j2}, a_{j3})q_k(T_2, T_3)$$

where q_h and q_k belong to some final group of states F_n but $a_{i3} \neq a_{j3}$ and change this instruction to the two following instructions

$$q_h(a_{i2}, a_{i3}) \rightarrow (a_{j2}, a_{j3})p_k(T_2, T_3)$$

$$p_k(a_{i2}, a_{j3}) \rightarrow (a_{j2}, a_{i3})q_k(T_2, T_3)$$

We do not change other instructions of M and in such a way, we obtain a simple inductive Turing machine K . As a result of these changes, if the state belongs to a final group, it always leaves this group when the output of K changes. So, the state stabilizes in some final group if and only if the output stabilizes. Consequently, the new inductive Turing machine K computes the given language L by state stabilizing. Theorem is proved. \square

Theorem 3.6 implies the following result.

Corollary 3.7. $L^{bt}(\text{ITM1}) \subseteq L^{st}(\text{ITM1})$.

This result shows that computation by output stabilizing looks more powerful than computation by bistabilizing in the class of all inductive Turing machines of the first order. However, Theorems 3.4 and 3.6 together imply that these modes are linguistically equivalent.

Theorem 3.7. *Computation by state stabilizing is linguistically equivalent to computation by bistabilizing in the class of all inductive Turing machines of the first order.*

Corollary 3.8. $L^{bt}(\text{ITM1}) = L^{st}(\text{ITM1})$.

Theorems 3.3 and 3.7 together imply that all considered computing modes are linguistically equivalent.

Theorem 3.8. *Computation by state stabilizing, computation by output stabilizing and computation by bistabilizing are linguistically equivalent in the class of all inductive Turing machines of the first order.*

Corollary 3.9. $L^{bt}(\text{ITM1}) = L^{st}(\text{ITM1}) = L^{ot}(\text{ITM1})$.

Inductive Turing machines allow modeling Turing machines, namely, for any Turing machine T , there is an inductive Turing machine M that has the same language as P (Burgin, 2005a). This makes possible to obtain the classical result of computability theory that computing by final state is linguistically equivalent to computing by halting (Burgin, 2005a; Hopcroft *et al.*, 2001; Sipser, 1996) as a direct corollary of Theorem 3.8.

Comparing the state stabilizing computed language $L^{st}(M)$ of an inductive Turing machine M , the output stabilizing computed language $L^{ot}(M)$ of the machine M and the bistabilizing computed language $L^{bt}(M)$ of the machine M , we see that in general these languages do not coincide. The same can be true for the accepted languages of the inductive Turing machine M . Such machines are considered in the following examples.

Indeed, the output stabilizing accepted language $L_{ot}(W)$ of the machine W from Example 3.2 is much larger than the bistabilizing accepted language $L_{bt}(W)$ of the same machine W . The same inequality is true for the computed languages of inductive Turing machine W . In addition, the state stabilizing accepted language $L_{st}(M)$ of the machine M is much larger than the bistabilizing accepted language $L_{bt}(M)$ of the same machine M . The same inequality is true for the computed languages of inductive Turing machine M .

However, Proposition 3.2 shows that for any Turing machine M , the language $L_{bt}(M)$ cannot be larger than the language $L_{st}(M)$ and the language $L_{bt}(M)$ cannot be larger than the language $L_{ot}(M)$.

In addition, for the classes of the output stabilizing accepted by inductive Turing machines of the first order languages and the state stabilizing accepted by inductive Turing machines of the first order languages this is not true.

Now let us explore for inductive Turing machines of the first order, relations between computed and accepted languages and classes of languages.

Theorem 3.9. *If a language L has an inductive Turing machine M of the first order that accepts L by state stabilizing, then L has an inductive Turing machine V of the first order that computes L by state stabilizing.*

Proof. Let us consider an inductive Turing machine M of the first order that accepts a language L by state stabilizing, i.e., $L = L_{st}(M)$. As it is proved that any an inductive Turing machine of the first order is equivalent to a simple inductive Turing machine M (Burgin, 2005a), it is possible to assume that M is a simple inductive Turing machine.

Then we change the rules R of the inductive Turing machine M to the rules P of the inductive Turing machine V by the following transformation. We exclude from all rules of the machine M any possibility to write something into the output tape. Besides, we add such rules according to which the process of functioning machine V begins so that the output head writes the input word into the output tape.

By construction, V is also a simple inductive Turing machine.

As there no other transformations of the system of initial rules, the output of V is never changing. By Definition 3.3, this output is the result of computation for V if and only if after some number of steps, the state of the machine M always remains in the same final group of states. It means that a word w is accepted by state stabilizing in the inductive Turing machine M if and only if the word w is computed by state stabilizing in the inductive Turing machine V . Consequently, languages $L_{st}(M)$ and $L^{st}(V)$ coincide. Theorem is proved. \square

Corollary 3.10. $L_{st}(\text{ITM1}) \subseteq L^{st}(\text{ITM1})$.

This result shows that computation by state stabilizing looks more powerful than acceptance by state stabilizing in the class of all inductive Turing machines of the first order.

Theorem 3.10. *If a language L has an inductive Turing machine M of the first order that computes L by output stabilizing, then there is an inductive Turing machine W of the first order that accepts L by output stabilizing.*

Proof. Let us consider an inductive Turing machine M of the first order that computes a language L by output stabilizing, i.e., $L = L^{ot}(M)$. As it is proved that any an inductive Turing machine of the first order is equivalent to a simple inductive Turing machine M (Burgin, 2005a), it is possible to assume that M is a simple inductive Turing machine.

In addition, we consider a Turing machine G that given a word 1^n , generates n different words in the alphabet X of the machine M , generating all words in the alphabet X in such a way. We also take a Turing machine C that compares its input with the word written in the tape of this machine and called the sample word of C . When both words are equal, the machine C gives 1 as its output and halts. Otherwise, the machine C gives 0 as its output and halts.

This allows us to build the machine W in the following way. It contains subroutines G_0 , C_0 and M_0 , that simulate the machines G , C and M , respectively. The machine W has as many working tapes as it is necessary for functioning of the subroutines G_0 , C_0 and M_0 . In particular, two counting tapes are added - the counting tape for the machine W and the counting tape for the subroutine M_0 . In these tapes, numbers of iterations are stored. Then we add rules for W such that allow it to perform the following steps.

1. When a word w comes to W as its input, the machine W writes w into the working tape of the machine C as its sample word and goes to the step 2.
2. The machine W writes the number 1 into the counter tape, which contains the number of iterations, and goes to the step 3.
3. The machine W gives the number 1 as the input to the subroutine G_0 and goes to the step 4.
4. The subroutine G_0 generates the word u_1 and gives it as the input to the subroutine M_0 , going to the step 5.
5. The subroutine M_0 computes with this input until the first word w_1 appears in the output tape of M_0 . Then the machine W goes to the step 6.
6. The machine W writes the word w_1 in its output tape and gives w_1 as the input to the subroutine C_0 , which compares w_1 and w . Then the machine W goes to the step 7 or 9 depending on the output of C_0 .
7. If the output of C_0 is 1, then the subroutine M_0 continues its computations until the next output word, in this case w_2 , appears in the output tape of M_0 . Then the machine W goes to the step 8.
8. The machine W gives w_2 as the input to the subroutine C_0 , which compares w_2 and w . Then the machine W goes to the step 7 or 9 depending on the output of C_0 . Note that if the output of C_0 is always 1 starting from some input, the machine W accepts w by output stabilizing.
9. If the output of C_0 is 0, then the machine W writes the word w in its output tape, changes this word to the word checked by C_0 (it will be w_1 if the previous step had number 6, while it will be w_2 if the previous step had number 8) and goes to the step 10.
10. The machine W adds 1 to the number in its counter tape, gives this new number n (in the second iteration $n = 2$) as the input to the subroutine G_0 and goes to the step 11.
11. The subroutine G_0 generates n words u_1, u_2, \dots, u_n and gives all of them one by one as the inputs to the subroutine M_0 .
12. The subroutine M_0 writes 1 into its counting tape and computes with the input u_1 until it writes n words into the output tape. Then the machine W goes to the step 13.
13. The machine W gives the current output word w_k of the subroutine M_0 as the input to the subroutine C_0 , which compares w_k and w . Then the machine W goes to the step 14 or 15 depending on the output of C_0 .
14. If the output of C_0 is 1, then the subroutine M_0 continues its computations until the next output word, say w_r , appears in the output tape of M_0 . Note that if the output of C_0 is always 1 starting from some input, the machine W accepts w by output stabilizing.

15. If the output of C_0 is 0 and the number t in its counting tape is less than k , then the machine W writes the word w in its output tape and changes this word to the word checked by C_0 , while the subroutine M_0 adds 1 to the number t in its counting tape and computes with the input u_{t+1} until it writes n words into the output tape. Then the machine W goes to the step 13.
16. If the number t in its counting tape of M_0 becomes equal to k , the machine W goes to the step 10.

If the word w is computed by output stabilizing of the inductive Turing machine M , then given some input u , the machine M makes some number of steps, and then its output becomes equal to w and stops changing. In this case, the output of C_0 is always 1 starting after some number of steps, and consequently, the machine W accepts w by output stabilizing.

If the word w is not computed by output stabilizing of the inductive Turing machine M for any input, then either the output of M is not stabilizing or it is stabilizing with the word v that is not equal to w . In the first case, the output of W is also not stabilizing and thus, the machine W does not accept w by output stabilizing. In the second case, the output of W is also not stabilizing because both words w and v appear infinitely many times in the output tape of W and thus, the machine W does not accept w by output stabilizing.

By construction, W is a simple inductive Turing machine.

Thus, the simple inductive Turing machine W does not accept w by output stabilizing is and only if the simple inductive Turing machine M does not accept w by output stabilizing.

Theorem is proved because w is an arbitrary word in the alphabet of the inductive Turing machine M . \square

Corollary 3.11. $L^{ot}(\text{ITM1}) \subseteq L_{ot}(\text{ITM1})$.

This result shows that acceptance by output stabilizing looks more powerful than computation by output stabilizing in the class of all inductive Turing machines of the first order.

Let us take an inductive Turing machine M functioning of which satisfies Condition ST and its state stabilizing language $L^{st}(M)$, i.e., the set of all words computed by state stabilizing of the machine M (cf. Definition 3.3). Thus, a word w is computed by state stabilizing of the machine M if and only if is computed by output stabilizing of the machine M . Consequently, $L^{st}(M) = L^{ot}(M)$. This gives us the following results.

Theorem 3.11. For any inductive Turing machine M of the first order functioning of which satisfies Condition ST, $L^{st}(M) \subseteq L^{ot}(M)$.

Corollary 3.12. $L^{st}(\text{ITM1}) \subseteq L^{ot}(\text{ITM1})$.

This result shows that computation by output stabilizing looks more powerful than computation by state stabilizing in the class of all inductive Turing machines of the first order.

Note that in general Theorem 11 does not imply equality of the classes $L^{st}(\text{ITM1})$ and $L^{ot}(\text{ITM1})$ because not all inductive Turing machines satisfy Condition ST.

At the same time, other results obtained in this paper allows us to find more exact relations between these classes of languages. Namely, by Corollary 3.1, we have $L_{st}(\text{ITM1}) \subseteq L_{ot}(\text{ITM1})$.

By Corollary 3.2, we have $\mathbf{L}^{ot}(\text{ITM1}) \subseteq \mathbf{L}^{bt}(\text{ITM1})$.

Comparing acceptance by state with acceptance by output, we obtain the following inclusion $\mathbf{L}_{ot}(\text{ITM1}) \subseteq \mathbf{L}_{st}(\text{ITM1})$.

By Corollary 3.4, we have $\mathbf{L}^{bt}(\text{ITM1}) \subseteq \mathbf{L}^{ot}(\text{ITM1})$.

By Corollary 3.10, we have $\mathbf{L}_{st}(\text{ITM1}) \subseteq \mathbf{L}^{st}(\text{ITM1})$.

By Corollary 3.11, we have $\mathbf{L}^{ot}(\text{ITM1}) \subseteq \mathbf{L}_{ot}(\text{ITM1})$.

By Corollary 3.12, we have $\mathbf{L}^{st}(\text{ITM1}) \subseteq \mathbf{L}^{ot}(\text{ITM1})$.

This gives us the following chain of inclusions:

$$\mathbf{L}^{st}(\text{ITM1}) \subseteq \mathbf{L}^{ot}(\text{ITM1}) \subseteq \mathbf{L}_{ot}(\text{ITM1}) \subseteq \mathbf{L}_{st}(\text{ITM1}) \subseteq \mathbf{L}^{st}(\text{ITM1}).$$

By properties of sets and the inclusion relation, the chain (5) implies that all inclusions in it are equalities. Thus, we have the following result.

Theorem 3.12. *In the class of all inductive Turing machines of the first order, the following modes of functioning are linguistically equivalent:*

1. *Acceptation by state stabilizing.*
2. *Acceptation by output stabilizing.*
3. *Computation by state stabilizing.*
4. *Computation by output stabilizing.*

Inductive Turing machines allow modeling Turing machines, namely, for any Turing machine T , there is an inductive Turing machine M that has the same language as P (Burgin, 2005a). This makes possible to obtain the following classical result of computability theory as a direct corollary of Theorem 3.12.

Proposition 3.4. *For Turing machines, the acceptance mode is linguistically equivalent to the computation mode, i.e., the class of languages accepted by Turing machines coincides with the class of languages computed by Turing machines.*

This result justifies the situation when in the majority of textbooks on computer science, it is assumed that Turing machines work in the acceptance mode and model computers and this allows modeling computers although computers, as a rule, work in the computation mode.

4. Conclusion

Various models of computation are studied in computer science - deterministic and nondeterministic finite automata, deterministic and nondeterministic pushdown automata, deterministic and nondeterministic Turing machines with one or many tapes, which can be one-dimensional and many-dimensional, and so on.

The basic results in this area are theorems on linguistic equivalence of different models of computation, i.e., equivalence with respect to the languages that are accepted/generated by these models. Thus, for finite automata, it is proved that the class of languages accepted by deterministic finite automata is the same as the class of languages accepted by nondeterministic finite automata

and as the class of languages accepted by nondeterministic finite automata with ε -transitions (cf., for example, (Hopcroft *et al.*, 2001): Theorem 2.12, Theorem 2.22; (Sipser, 1996): Theorem 1.19)).

In the theory of pushdown automata, it is proved that the class of languages accepted by pushdown automata by final state is the same as the class of languages accepted by pushdown automata by empty stack and as the class of languages generated by context free grammars (cf., for example, (Hopcroft *et al.*, 2001): Theorem 6.9, Theorem 6.11, Theorem 6.14; (Sipser, 1996): Theorem 2.12).

In the theory of Turing machines, it is proved that the class of languages accepted by deterministic Turing machines with a single tape is the same as the class of languages accepted by nondeterministic Turing machines with a single and as the class of languages accepted by Turing machines with many tapes and as the class of languages accepted by Turing machines with multidimensional tapes and as the class of languages accepted by pushdown automata with two and more stacks (cf., for example, (Hopcroft *et al.*, 2001): Theorem 8.9, Theorem 8.11, Theorem 6.14, Theorem 8.13; (Sipser, 1996): Theorem 3.8, Theorem 3.10).

In this paper, we obtained similar results for inductive Turing machines. Namely, it is proved that: (1) the class of languages computed by output stabilizing of inductive Turing machines of the first order is the same as the class of languages computed by bistabilizing of inductive Turing machines of the first order (Theorem 3.3); (2) the class of languages computed by output stabilizing of inductive Turing machines of the first order is the same as the class of languages computed by state stabilizing of inductive Turing machines of the first order (Theorem 3.8); (2) the class of languages computed by output (state) stabilizing of inductive Turing machines of the first order is the same as the class of languages accepted by output (state) stabilizing of inductive Turing machines of the first order (Theorem 3.12).

It is necessary to remark that it is possible to include the results of this paper into a standard course of the theory of automata, formal languages and computation.

The obtained results bring us to the following problems.

Problem 1. Study different modes of functioning for inductive Turing machines of the higher orders.

Problem 2. Study inductive Turing machines in which the control device is a more powerful automaton than a finite automaton.

Here we considered accepting and computing modes of inductive Turing machine brings us to the following problem.

Problem 3. For inductive Turing machines, study relations between the decision mode, accepting mode and computing mode of functioning.

It would be important to study properties of stabilizing computations in distributed systems. Grid automata provide the most advanced and general model of distributed systems (Burgin, 2005a).

Problem 4. Study properties of stabilizing computations for grid automata.

There are models of computation without explicit utilization of automata (cf., for example, (Milner, 1989; Lee & Sangiovanni-Vincentelli, 1996; Burgin & Smith, 2010)).

Problem 5. Study properties of stabilizing computations utilizing models of concurrent computational processes.

References

- Beros, A. A. (2013). Learning theory in the arithmetical hierarchy, preprint in mathematics. *math.LO/1302.7069* (electronic edition: <http://arXiv.org>).
- Büchi, J. Richard (1960). Weak second-order arithmetic and finite automata. *Z. Math. Logik und Grundl. Math.* **6**, 66–92.
- Burgin, M. (1999). Super-recursive algorithms as a tool for high performance computing. *Proceedings of the High Performance Computing Symposium, San Diego* **6**, 224–228.
- Burgin, M. (2003). Nonlinear phenomena in spaces of algorithms. *International Journal of Computer Mathematics* **80**(12), 1449–1476.
- Burgin, M. (2004). Algorithmic complexity of recursive and inductive algorithms. *Theoretical Computer Science* **317**(13), 31 – 60.
- Burgin, M. (2005a). *Super-recursive Algorithms*. Springer, New York.
- Burgin, M. (2005b). Superrecursive hierarchies of algorithmic problems. In: *Proceedings of the 2005 International Conference on Foundations of Computer Science, CSREA Press, Las Vegas*. pp. 31–37.
- Burgin, M. (2006). Algorithmic control in concurrent computations. *Proceedings of the 2006 International Conference on Foundations of Computer Science, CSREA Press, Las Vegas* **6**, 17–23.
- Burgin, M. (2007). Algorithmic complexity as a criterion of unsolvability. *Theoretical Computer Science* **383**(2/3), 244 – 259.
- Burgin, M. (2010a). Algorithmic complexity of computational problems. *International Journal of Computing & Information Technology*. **2**(1), 149–187.
- Burgin, M. (2010b). *Measuring power of algorithms, computer programs, and information automata*. Nova Science Publishers, New York.
- Burgin, M. and A. Klinger (2004). Experience, generations, and limits in machine learning. *Theoretical Computer Science* **317**(1/3), 71 – 91.
- Burgin, M. and B. Gupta (2012). Second-level algorithms, superrecursivity, and recovery problem in distributed systems. *Theory of Computing Systems* **50**(4), 694 – 705.
- Burgin, M. and E. Eberbach (2008). Cooperative combinatorial optimization: Evolutionary computation case study. *Biosystems* **91**(1), 34 – 50.
- Burgin, M. and E. Eberbach (2009a). On foundations of evolutionary computation: An evolutionary automata approach. *Handbook of Research on Artificial Immune Systems and Natural Computing: Applying Complex Adaptive Technologies*, Hongwei Mo, Ed., IGI Global, Hershey, Pennsylvania, pp. 342–360.
- Burgin, M. and E. Eberbach (2009b). Universality for Turing machines, inductive Turing machines and evolutionary algorithms. *Fundamenta Informaticae* **91**, 53–77.
- Burgin, M. and E. Eberbach (2010). Bounded and periodic evolutionary machines. *Proc. 2010 Congress on Evolutionary Computation (CEC'2010), Barcelona, Spain* pp. 1379–1386.
- Burgin, M. and E. Eberbach (2012). Evolutionary automata: Expressiveness and convergence of evolutionary computation. *The Computer Journal* **55**(9), 1023–1029.
- Burgin, M. and M. L. Smith (2010). A theoretical model for grid, cluster and internet computing. *Selected Topics in Communication Networks and Distributed Systems*, World Scientific, New York/London/Singapore pp. 485 – 535.
- Burgin, M. and N. Debnath (2004). Measuring software maintenance. *Proceedings of the ISCA 19th International Conference "Computers and their Applications", ISCA, Seattle, Washington* pp. 118–121.
- Burgin, M. and N. Debnath (2005). Complexity measures for software engineering. *J. Comp. Methods in Sci. and Eng.* **5**(1 Supplement), 127–143.
- Burgin, M. and N. Debnath (2009). Super-recursive algorithms in testing distributed systems. *Proceedings of the ISCA 24-th International Conference "Computers and their Applications", ISCA, New Orleans, Louisiana, USA* pp. 209–214.

- Burgin, M., N. Debnath and H. K. Lee (2009). Measuring testing as a distributed component of the software life cycle. *Journal of Computational Methods in Science and Engineering* **9**(Supplement 2/ 2009), 211–223.
- Burks, A. W. and J. B. Wright (1953). Theory of logical nets. *Proceedings of the IRE* **41**(10), 1357–1365.
- Calinescu, R., R. France and C. Ghezzi (2013). Editorial. *Computing* **95**(3), 165–166.
- Chadha, R., A. P. Sistla and M. Viswanathan (2009). Power of randomization in automata on infinite strings. In: *CONCUR 2009 - Concurrency Theory* (Mario Bravetti and Gianluigi Zavattaro, Eds.). Vol. 5710 of *Lecture Notes in Computer Science*. pp. 229–243. Springer, Berlin/Heidelberg.
- Gold, E. M. (1967). Language identification in the limit. *Information and Control* **10**(5), 447 – 474.
- Hopcroft, J. E., R. Motwani and J. D. Ullman (2001). *Introduction to Automata Theory, Languages, and Computation*. Addison Wesley, Boston/San Francisco/New York.
- Lee, E.A. and A. Sangiovanni-Vincentelli (1996). Comparing models of computation. In: *Computer-Aided Design, 1996. ICCAD-96. Digest of Technical Papers., 1996 IEEE/ACM International Conference on*. pp. 234–241.
- Milner, R. (1989). *Communication and Concurrency*. Prentice-Hall, Inc.. Upper Saddle River, NJ, USA.
- Rogers, Jr., Hartley (1987). *Theory of Recursive Functions and Effective Computability*. MIT Press. Cambridge, MA, USA.
- Sipser, M. (1996). *Introduction to the Theory of Computation*. 1st ed.. International Thomson Publishing.
- Thomas, Wolfgang (1990). Handbook of theoretical computer science (vol. b). Chap. Automata on Infinite Objects, pp. 133–191. MIT Press. Cambridge, MA, USA.



Second Hankel Determinant for Generalized Sakaguchi Type Functions

S. P. Vijayalakshmi^{a,*}, T. V. Sudharsan^b

^a*Department of Mathematics, Ethiraj College, Chennai - 600 008, India*

^b*Department of Mathematics, SIVET College, Chennai - 600 073, India*

Abstract

In this paper, we have obtained sharp upper bounds for the functional $|a_2a_4 - a_3^2|$ belonging to a new subclass of generalized Sakaguchi type functions introduced by (Frasin, 2010).

Keywords: Sakaguchi functions, subordination, Hankel determinant, starlike functions.

2010 MSC: 30C45, 30C50.

1. Introduction

Let A be the class of analytic functions of the form

$$f(z) = z + \sum_{n=2}^{\infty} a_n z^n; \quad (z \in \Delta := \{z \in \mathbb{C} : |z| < 1\}) \quad (1.1)$$

and S be the subclass of A consisting of univalent functions. For two functions $f, g \in A$, we say that the function $f(z)$ is subordinate to $g(z)$ in Δ and write $f < g$, or $f(z) < g(z)$; ($z \in \Delta$) if there exists an analytic function $w(z)$ with $w(0) = 0$ and $|w(z)| < 1$ ($z \in \Delta$), such that $f(z) = g(w(z))$, ($z \in \Delta$). In particular, if the function g is univalent in Δ , the above subordination is equivalent to $f(0) = g(0)$ and $f(\Delta) \subset g(\Delta)$.

Denote by S_S^* the subclass of S consisting of functions given by (1.1) satisfying $\operatorname{Re} \left[\frac{zf'(z)}{f(z)-f(-z)} \right] > 0$ for $z \in \Delta$. These functions introduced by (Sakaguchi, 1959) are called functions starlike with respect to symmetric points.

*Corresponding author

Email addresses: vijishreekanth@gmail.com (S. P. Vijayalakshmi), tvsudharsan@rediffmail.com (T. V. Sudharsan)

Recently (Frasin, 2010) introduced and studied a generalized Sakaguchi type class $S(\alpha, s, t)$ if it satisfies

$$\operatorname{Re} \left\{ \frac{(s-t)zf'(z)}{f(sz) - f(tz)} \right\} > \alpha \quad (1.2)$$

for some $0 \leq \alpha < 1$, $s, t \in C$ with $s \neq t$ and for all $z \in \Delta$. Also denote by $T(\alpha, s, t)$ the subclass of A consisting of all functions $f(z)$ such that $zf'(z) \in S(\alpha, s, t)$. The class $S(\alpha, 1, t)$ was introduced and studied by Owa et al. (Owa et al., 2005, 2007). If $t = -1$, the class $S(\alpha, 1, -1) \equiv S_s(\alpha)$ (Sakaguchi, 1959) is called Sakaguchi function of order α (see (Cho et al., 1993; Owa et al., 2005)), where as $S_s(0) = S_s^*$ (Sakaguchi, 1959).

Note that $S(\alpha, 1, 0) \equiv S^*(\alpha)$ and $T(\alpha, 1, 0) \equiv C(\alpha)$ which are, respectively, the familiar classes of starlike functions of order α ($0 \leq \alpha < 1$) and convex functions of order α ($0 \leq \alpha < 1$).

Mathur & Mathur (Trilok Mathur & Ruchi Mathur, 2012) investigated the classes $S_s^*(\phi, s, t)$ and $T(\phi, s, t)$ defined as follows.

Definition 1.1. Let $\phi(z) = 1 + B_1z + B_2z^2 + \dots$ be univalent starlike function with respect to 1 which maps the unit disk Δ onto a region in the right half plane which is symmetric with respect to the real axis, and let $B_1 > 0$. The function $f \in A$ is in the class $S_s^*(\phi, s, t)$ if

$$\left\{ \frac{(s-t)zf'(z)}{f(sz) - f(tz)} \right\} < \phi(z), \quad s \neq t.$$

Remark. $T(\phi, s, t)$ denotes the subclass of A consisting functions $f(z)$ such that $zf'(z) \in S_s^*(\phi, s, t)$. Observe that $S_s^*(\phi, 1, 0) \equiv S_s^*(\phi)$ and $T(\phi, 1, 0) \equiv C(\phi)$, which are the classes introduced and studied by Ma and Minda (Ma & Minda, 1994). Also note that $S_s^*(\phi, 1, -1) \equiv S_s^*(\phi)$, Shanmugam et al. (Shanmugam et al., 2006).

The q^{th} Hankel determinant for $q \geq 1$ and $n \geq 0$ is stated by Noonan and Thomas (Noonan & Thomas, 1976) as

$$H_q(n) = \begin{vmatrix} a_n & a_{n+1} & \cdots & a_{n+q+1} \\ a_{n+1} & \cdots & & \cdots \\ \vdots & & & \vdots \\ a_{n+q-1} & \cdots & & a_{n+2q-2} \end{vmatrix}$$

This determinant has also been considered by several authors. For example, Noor (Noor, 1983) determined the rate of growth of $H_q(n)$ as $n \rightarrow \infty$ for functions f given by (1.1) with bounded boundary. In particular, sharp upper bounds on $H_2(2)$ were obtained by the authors of articles (Hayami & Owa, 2010; Janteng et al., 2008; Kharudin et al., 2011; Noor, 1983; Selvaraj & Vasanthi, 2010) for different classes of functions.

Easily, one can observe that the Fekete-Szego functional is $H_2(1)$. Fekete-Szego then further generalised the estimate $|a_3 - \mu a_2^2|$ where μ is real and $f \in S$. For our discussion in this paper, we consider the Hankel determinant in the case $q = 2$ and $n = 2$,

$$H_2(2) = \begin{vmatrix} a_2 & a_3 \\ a_3 & a_4 \end{vmatrix}.$$

In the present investigation, we see the upper bound for the functional $|a_2a_4 - a_3^2|$ belonging to a new subclass $S_s^*(\phi, s, t)$ of generalized Sakaguchi type functions introduced by (Frasin, 2010).

2. Preliminary Results

Let P denote the class of functions

$$p(z) = 1 + c_1z + c_2z^2 + \cdots \quad (2.1)$$

which are regular in Δ and satisfy $\operatorname{Re} p(z) > 0$, $z \in \Delta$. Throughout this paper we assume that $p(z)$ is given by (2.1) and $f(z)$ is given by (1.1). To prove the main results we shall require the following lemmas.

Lemma 2.1. (Duren, 1983) Let $p \in P$, then $|c_k| \leq 2$, $k = 1, 2, \dots$ and the inequality is sharp.

Lemma 2.2. (Libera & Zlotkiewicz, 1982, 1983) Let $p \in P$, then

$$2c_2 = c_1^2 + x(4 - c_1^2) \quad (2.2)$$

and

$$4c_3 = c_1^3 + 2xc_1(4 - c_1^2) - x^2c_1(4 - c_1^2) + 2y(1 - |x|^2)(4 - c_1^2) \quad (2.3)$$

for some x, y such that $|x| \leq 1$ and $|y| \leq 1$.

3. Main Results

Theorem 3.1. If $f \in S_S^*(\phi, s, t)$, then

$$|a_2a_4 - a_3^2| \leq \frac{B_1}{(3 - s^2 - st - t^2)^2}, \text{ provided } s + t \neq 2. \quad (3.1)$$

The result obtained is sharp.

Proof. Let $f \in S_S^*(\phi, s, t)$. Then there exists a Schwarz function $w(z) \in A$ such that

$$\left\{ \frac{(s-t)zf'(z)}{f(sz) - f(tz)} \right\} = \phi(w(z)), \quad (z \in \Delta, s \neq t) \quad (3.2)$$

If $P_1(z)$ is analytic and has positive real part in Δ and $P_1(0) = 1$, then

$$P_1(z) = \frac{1 + w(z)}{1 - w(z)} = 1 + c_1z + c_2z^2 + \cdots \quad (3.3)$$

From (3.3) we obtain

$$w(z) = \frac{c_1}{2}z + \frac{1}{2}\left(c_2 - \frac{c_1^2}{2}\right)z^2 + \cdots \quad (3.4)$$

Let

$$p(z) = \frac{(s-t)zf'(z)}{f(sz) - f(tz)} = 1 + b_1z + b_2z^2 + \cdots; \quad (z \in \Delta) \quad (3.5)$$

which gives

$$\begin{aligned} b_1 &= (2 - s - t)a_2, \\ b_2 &= (3 - s^2 - st - t^2)a_3 + (s + t)(s + t - 2)a_2^2, \\ b_3 &= (4 - s^3 - st^2 - s^2t - t^3)a_4 + 2(s^2 + st + t^2)(s + t - 1)a_2a_3 \\ &\quad + 2a_2^3(s + t)^2 - 3a_2a_3(s + t). \end{aligned}$$

Since $\phi(z)$ is univalent and $P < \phi$, therefore using (3.4) we obtain

$$\begin{aligned} P(z) = \phi(w(z)) &= 1 + \frac{B_1c_1}{2}z + \left\{ \frac{1}{2} \left(c_2 - \frac{c_1^2}{2} \right) B_1 + \frac{1}{4}c_1^2B_2 \right\} z^2 \\ &\quad + \left[\frac{B_1}{2} \left\{ 2c_3 + c_1 \left(\frac{c_1^2}{2} - c_2 \right) - c_1c_2 \right\} \right. \\ &\quad \left. + \frac{B_1c_1}{2} \left(c_2 - \frac{c_1^2}{2} \right) + \frac{B_3c_1^3}{8} \right] z^3 + \dots \end{aligned} \quad (3.6)$$

Now from (3.5) and (3.6) we have

$$\frac{(s-t)zf'(z)}{f(sz) - f(tz)} = 1 + \frac{B_1c_1}{2}z + \left\{ \frac{1}{2} \left(c_2 - \frac{c_1^2}{2} \right) B_1 + \frac{1}{4}c_1^2B_2 \right\} z^2 + \dots \quad (3.7)$$

On equating the coefficient of z , z^2 and z^3 in (3.7) we obtain

$$\begin{aligned} a_2 &= \frac{B_1c_1}{2(2-s-t)}, \\ a_3 &= \frac{1}{(3-s^2-st-t^2)} \left\{ \frac{1}{2} \left(c_2 - \frac{c_1^2}{2} \right) B_1 + \frac{1}{4}c_1^2B_2 + \frac{(s+t)B_1^2c_1^2}{4(2-s-t)} \right\}, \\ a_4 &= \frac{1}{(4-s^3-st^2-s^2t-t^3)} \left[B_1c_3 + c_1^3 \left[\frac{B_1}{4} - \frac{B_2}{4} + \frac{B_3}{8} - \frac{B_1^3(s+t)^2}{4(2-s-t)^3} \right. \right. \\ &\quad \left. \left. + \frac{(s+t+2(s^2+st+t^2))}{16(2-s-t)(3-s^2-st-t^2)} \left\{ B_1B_2 - B_1^2 + \frac{B_1^3}{(2-s-t)} \right\} \right] \right. \\ &\quad \left. + c_1c_2 \left[-\frac{3B_1}{2} + \frac{B_2}{2} \right. \right. \\ &\quad \left. \left. + \frac{B_1^2}{8(2-s-t)(3-s^2-st-t^2)} [3(s+t) - 2(s^2+st+t^2)(s+t-1)] \right] \right]. \end{aligned}$$

Thus we have,

$$\begin{aligned}
 |a_2a_4 - a_3^2| = & \left| \frac{B_1c_1}{2P_1} \left\{ B_1c_3 + c_1^3 \left[\frac{2(B_1 - B_2) + B_3}{8} - \frac{B_1^3(s+t)^2}{4(2-s-t)^3} \right. \right. \right. \\
 & + \frac{3(s+t) - 2(s^2 + st + t^2)(s+t-1)}{16(2-s-t)(3-s^2-st-t^2)} \left. \left. \left\{ B_1B_2 - B_1^2 + \frac{B_1^3}{(2-s-t)} \right\} \right] \right\} \\
 & + c_1c_2 \left[-\frac{3B_1}{2} + \frac{B_2}{2} + \frac{B_1^2}{8(2-s-t)(3-s^2-st-t^2)} \right. \\
 & \left. \left. [3(s+t) - 2(s^2 + st + t^2)(s+t-1)] \right] \right. \\
 & \left. - \frac{1}{P_3} \left\{ c_1^4f_1(s,t) + c_1^2g_1(s,t) + \frac{c_1^2B_1^2}{16} + \frac{c_2^2B_1^2}{4} \right\} \right|.
 \end{aligned}$$

Suppose now that $c_1 = c$. Since $|c| = |c_1| \leq 2$, using the Lemma 2.1, we may assume without restriction $c \in [0, 2]$. Substituting for c_2 and c_3 , from Lemma 2.2 and applying the triangle inequality with $\rho = |x|$, we obtain

$$\begin{aligned}
 |a_2a_4 - a_3^2| \leq & c^4 \left[\frac{f(s,t)}{P_1} + \frac{B_1g(s,t)}{4P_1} - \frac{f_1(s,t)}{P_3} - \frac{g_1(s,t)}{2P_3} - \frac{B_1}{16P_3} \right] \\
 & + c^2(4-c^2)\rho \left[\frac{B_1}{4P_1} + \frac{B_1^2}{4P_1} - \frac{B_1^2\rho}{8P_1} - \frac{g_1(s,t)}{2P_3} - \frac{B_1}{8P_3} \right] \\
 & + \frac{c^2B_1^2}{16P_3} + \frac{B_1^2c(4-c^2)(1-\rho^2)}{4P_1} + \frac{B_1\rho^2(4-c^2)^2}{16P_3} \\
 = & F(\rho)
 \end{aligned} \tag{3.8}$$

where,

$$\begin{aligned}
 P_1 &= (2-s-t)(4-s^3-st^2-s^2t-t^3), \\
 P_2 &= 8(2-s-t)(3-s^2-st-t^2), \\
 P_3 &= (3-s^2-st-t^2)^2,
 \end{aligned}$$

$$\begin{aligned}
 f_1(s,t) &= \frac{B_2^2}{16} + \frac{(s+t)^2B_1^4}{16(2-s-t)^2} - \frac{B_1B_2}{8} + \frac{B_1^2B_2(s+t)}{8(2-s-t)} + \frac{(s+t)B_1^3}{8(2-s-t)}, \\
 g_1(s,t) &= \frac{(s+t)B_1^3}{4(2-s-t)} + \frac{B_1B_2}{4} - \frac{B_1^2}{4}, \\
 f(s,t) &= \frac{B_1^2}{4} - \frac{B_1B_2}{8} + \frac{B_1B_3}{8} - \frac{B_1^4(s+t)^2}{8(2-s-t)^3} \\
 &+ \frac{[(B_1B_2 - B_1^2)(2-s-t)] + B_1^3[3(s+t) - 2(s^2 + st + t^2)(s+t-1)]}{32(2-s-t)^2(3-s^2-st-t^2)}, \\
 g(s,t) &= -B_1 + \frac{B_2}{2} - \frac{B_1^2[3(s+t) - 2(s^2 + st + t^2)(s+t-1)]}{2},
 \end{aligned}$$

with $\rho = |x| \leq 1$. Furthermore

$$F'(\rho) = c^2(4 - c^2) \left[\frac{B_1}{4P_1} + \frac{B_1^2}{4P_1} - \frac{B_1^2\rho}{8P_1} - \frac{g_1(s, t)}{2P_3} - \frac{B_1}{8P_3} \right] \\ + \frac{B_1^2c\rho(4 - c^2)(4 - c)}{8P_1} + \frac{B_1^2(4 - c^2)^2\rho}{8P_3}.$$

For a $c \in [0, 2]$, $F(\rho) \leq F(1)$, that is

$$|a_2a_4 - a_3^2| \leq c^4 \left[\frac{f(s, t)}{P_1} + \frac{B_1g(s, t)}{4P_1} - \frac{f_1(s, t)}{P_3} - \frac{g_1(s, t)}{2P_3} - \frac{B_1}{16P_3} \right] \\ + c^2(4 - c^2) \left[\frac{B_1}{4P_1} + \frac{B_1^2}{4P_1} - \frac{B_1^2}{8P_1} - \frac{g_1(s, t)}{2P_3} - \frac{B_1}{8P_3} \right] \\ + \frac{c^2B_1^2}{16P_3} + \frac{B_1(4 - c^2)^2}{16P_3} \\ = G(c).$$

By elementary calculus we have $G''(c) \leq 0$ for $0 \leq c \leq 2$ and $G(c)$ has real critical point at $c = 0$. Thus the upper bound of $F(\rho)$ corresponds to $\rho = 1$ and $c = 0$. Thus the maximum of $G(c)$ occurs at $c = 0$. Hence,

$$|a_2a_4 - a_3^2| \leq \frac{B_1}{(3 - s^2 - st - t^2)^2}$$

If $p(z) \in P$ with $c_1 = c = 0$, $c_2 = 2$, $c_3 = 1$, then we obtain

$p(z) = (1 - z) + \frac{z}{(1-z)^2} = 1 + 2z^2 + z^3 + \dots \in P$. The result is sharp for the functions defined by

$$\left\{ \frac{(s - t)zf'(z)}{f(sz) - f(tz)} \right\} = \phi(z), \quad s \neq t$$

and

$$\left\{ \frac{(s - t)zf'(z)}{f(sz) - f(tz)} \right\} = \phi(z^2), \quad s \neq t.$$

□

Remark. If $f \in S_S^*(\phi, 1, -1)$, then

$$|a_2a_4 - a_3^2| \leq \frac{B_1}{2}.$$

Since $f(z) \in T(\phi, s, t)$ if and only if $zf'(z) \in S_S^*(\phi, s, t)$, proceeding on similar lines as in Theorem 3.1 we obtain the upper bound for the functional $|a_2a_4 - a_3^2|$ belonging to the class $T(\phi, s, t)$ which is stated below without proof.

Theorem 3.2. If $f \in T(\phi, s, t)$, then

$$|a_2a_4 - a_3^2| \leq \frac{B_1^2}{9(3 - s^2 - st - t^2)^2}, \quad \text{provided } s + t \neq 2. \quad (3.9)$$

The result obtained is sharp.

Remark. If $f \in T(\phi, 1, -1)$, then

$$|a_2a_4 - a_3^2| \leq \frac{B_1^2}{36}.$$

Acknowledgement

The authors thank the referee for the valuable comments and suggestions to improve the presentation of the paper.

References

- Cho, N. E., O. S. Kwon and S. Owa (1993). Certain subclasses of Sakaguchi functions. *SEA Bull. Math.* **17**, 121–126.
- Duren, P. L. (1983). *Univalent functions*. Springer Verlag, New York Inc.
- Frasin, B. A. (2010). Coefficient inequalities for certain classes of Sakaguchi type functions. *Int. J. Nonlinear Sci.* **10**(2), 206–211.
- Hayami, T. and S. Owa (2010). Generalized Hankel determinant for certain classes. *Int. Journal of Math. Analysis* **4**(52), 2473–2585.
- Janteng, A., S. A. Halim and M. Darus (2008). Estimate on the second Hankel functional for functions whose derivative has a positive real part. *Journal of Quality Measurement and Analysis* **4**(1), 189–195.
- Kharudin, N., A. Akbarally, D. Mohamad and S. C. Soh (2011). The second Hankel determinant for the class of close to convex functions. *European Journal of Scientific Research* **66**(3), 421–427.
- Libera, R. J. and E. J. Zlotkiewicz (1982). Early coefficients of the inverse of a regular convex function. *Proc. Amer. Math. Soc.* **85**(2), 225–230.
- Libera, R. J. and E. J. Zlotkiewicz (1983). Coefficient bounds for the inverse of a function with derivative in p . *Proc. Amer. Math. Soc.* **87**(2), 251–257.
- Ma, W. and D. Minda (1994). A unified treatment of some special classes of univalent functions. In: *Proceedings of Conference of Complex Analysis* (Z. Li, F. Ren, L. Yang and S. Zhang, Eds.). Intenational Press. pp. 157–169.
- Noonan, J. W. and D. K. Thomas (1976). On the second Hankel determinant of areally mean p -valent functions. *Transactions of the Americal Mathematical Society* **223**(2), 337–346.
- Noor, K. I. (1983). Hankel determinant problem for the class of functions with bounded boundary rotation. *Rev. Roum. Math. Pures Et Appl.* **28**(c), 731–739.
- Owa, S., T. Sekine and R. Yamakawa (2005). Notes on Sakaguchi type functions. *RIMS Kokyuroku* **1414**, 76–82.
- Owa, S., T. Sekine and R. Yamakawa (2007). On Sakaguchi type functions. *Appl Math. Comput.* **187**, 356–361.
- Sakaguchi, K. (1959). On a certain univalent mapping. *J. Math. Soc. Japan* **11**, 72–75.
- Selvaraj, C. and N. Vasanthi (2010). Coefficient bounds for certain subclasses of close-to-convex functions. *Int. Journal of Math. Analysis* **4**(37), 1807–1814.
- Shanmugham, T. N., C. Ramachandran and V. Ravichandran (2006). Fekete-Szegő problem for a subclasses of starlike functions with respect to symmetric points. *Bull. Korean Math. Soc.* **43**(3), 589–598.
- Trilok Mathur and Ruchi Mathur (2012). Fekete-Szegő inequalities for generalized Sakaguchi type functions. In: *Proceedings of the World Congress on Engineering, WCE 2012*. Vol. 1. London, U.K.



L_p - Approximation of Analytic Functions on Compact Sets Bounded by Jordan Curves

Devendra Kumar^{a,*}, Vandna Jain^b

^aDepartment of Mathematics, M.M.H. College, Ghaziabad-201 001, U.P. India

^bDepartment of Mathematics, Punjab Technical University, Jalandhar (Pb.), India

Abstract

This paper is concerned with functions analytic on compact sets bounded by Jordan curves having rapidly increasing maximum modulus such that order of function is infinite. To study the precise rates of growth of such functions the concept of index has been used. The q -order and lower q -order of analytic functions have been obtained in terms of L_p -approximation error. Our results improve and refine the results of Andre Giroux (Giroux, 1980) and Kapoor and Nautiyal (Kapoor & Nautiyal, 1982) for non entire case.

Keywords: L_p -approximation error, index- q , transfinite diameter, Faber series.

2010 MSC: Primary 30D10; Secondary 41A10.

1. Introduction

Let D be a compact set containing at least two points such that its complement D' with respect to the extended complex plane is a simply connected domain containing the point at infinity. In view of Riemann mapping theorem, there exists a one-one analytic function $z = \varphi(w)$ which maps $\{w : |w| > 1\}$ conformally onto D' such that $\varphi(\infty) = \infty$ and $\varphi'(\infty) > 0$. Thus, in a neighborhood of infinity, the function has the expansion

$$z = \varphi(w) = d \left[w + d_0 + \frac{d_{-1}}{w} + \dots \right]$$

where the number $d > 0$ is called the transfinite diameter of D . If we define $\eta(w) = \varphi(w/d)$, then η maps $\{w : |w| > d\}$ onto D' in a one-one conformal manner. If $w = \Omega(z)$ is the inverse function of η then $\Omega(\infty) = \infty$ and $\lim_{z \rightarrow \infty} \Omega(z)/z = 1$.

*Corresponding author

Email addresses: d_kumar001@rediffmail.com (Devendra Kumar), vandnajain.mittal@gmail.com (Vandna Jain)

For $1 \leq p < \infty$, let $L_p(D)$ denote the space of analytic functions f in D such that

$$\|f\|_{D,p} = \left(\frac{1}{A} \int_D |f(z)|^p dx dy \right)^{1/p} < \infty, \text{ where } A \text{ is the area of } D.$$

Let L_r is an analytic Jordan curve for each $r > d$. If D_r denotes the domain bounded by L_r , then $D \subset D_r$ for each $r > d$. Let $H(\overline{D}; R)$ denotes the class of all functions that are regular in D_R with a singularity on L_R ($d < R < \infty$). Since $\overline{D} \subset D_R$ for $R > d$ it follows that every $f \in H(\overline{D}; R)$ is analytic in \overline{D} and so $\int_D |f(z)|^p dx dy < \infty$ and $f \in L_p(D)$.

We now prove the following:

Theorem 1.1. Every $f \in H(\overline{D}; R)$ can be represented by the Faber series

$$f(z) = \sum_{n=0}^{\infty} a_n P_n(z), \quad z \in D_R \quad (1.1)$$

with

$$a_n = \frac{1}{2\pi i} \int_{|\xi|=1} f(\eta(\xi)) \xi^{-n-1} d\xi, \quad d < r < R \quad (1.2)$$

if and only if

$$\limsup_{n \rightarrow \infty} |a_n|^{1/n} = \frac{1}{R}. \quad (1.3)$$

The series in (1.1) converges absolutely and uniformly on every compact subset of D_R and diverge outside L_R .

Proof. Let $f \in H(\overline{D}; R)$. If $z \in D_R$, then $z \in D_r$ for some r satisfying $d < r < R$. Using Cauchy's integral formula,

$$f(z) = \frac{1}{2\pi i} \int_{L_r} \frac{f(t) dt}{t - z} = \frac{1}{2\pi i} \int_{|\xi|=r} \frac{f(\eta(\xi)) \eta'(\xi)}{n(\xi) - z} d\xi = \frac{1}{2\pi i} \int_{|\xi|=r} \left(\sum_{n=0}^{\infty} \frac{f(\eta(\xi))}{\xi^{n+1}} P_n(z) \right) d\xi.$$

Since the series under the integral sign converges uniformly on $|\xi| = r$, it can be integrated term by term. Thus we have

$$f(z) = \sum_{n=0}^{\infty} a_n P_n(z), \quad z \in D_R.$$

If

$$\overline{M}(r, f) = \max_{|\xi|=r} |f(\eta(\xi))|,$$

then (1.2) gives

$$|a_n| \leq \frac{\overline{M}(r, f)}{r^n}, \quad n = 0, 1, 2, \dots, \quad (1.4)$$

which are analogous of Cauchy's inequality for Taylor series. From (1.4), we have

$$\limsup_{n \rightarrow \infty} |a_n|^{1/n} \leq \frac{1}{r}.$$

Since this holds for every r satisfying $d < r < R$, we have

$$\limsup_{n \rightarrow \infty} |a_n|^{1/n} \leq \frac{1}{R}.$$

We now show that inequality does not holds in the above relation. If

$$\limsup_{n \rightarrow \infty} |a_n|^{1/n} = \frac{1}{R_0} < \frac{1}{R},$$

let r satisfy $d < r < R_0$; then for every ε such that $0 < \varepsilon < R_0 - r$, we have

$$|a_n| < \frac{1}{(R_0 - \varepsilon/2)^n} < \frac{1}{(r + \varepsilon/2)^n} \text{ for } n \geq n_0.$$

On the other hand, for every $z \in L_r$, we have $|P_n(z)| < (r + \varepsilon/4)^n$ for $n \geq n_1$.

Thus, for all $z \in L_r$, $|a_n P_n(z)| < \left(\frac{r + \varepsilon/4}{r + \varepsilon/2}\right)^n < 1$ for $n \geq \max(n_0, n_1)$.

The above inequality shows that the series $\sum_{n=0}^{\infty} a_n P_n(z)$ converges uniformly on L_r and hence on D_r . Since this is true for every r satisfying $d < r < R_0$, it follows that the series $\sum_{n=0}^{\infty} a_n P_n(z)$ converges uniformly on every compact subset of D_{R_0} to a function, $F(z)$, say. The function $F(z)$ must be regular in D_{R_0} since each term of the series is a regular function. However, on $D_R \subset D_{R_0}$, the series converges to $f(z)$ so that $F(z)$ is the analytic continuation of $f(z)$ to D_{R_0} . Since this contradicts the hypothesis that f has a singular point on L_R ($R < R_0$), we must have (1.3).

To show that the series (1.3) diverges outside L_R , let z_0 lie outside L_R . Then we have $|\Omega(z_0)| > R$. In view of $\lim_{n \rightarrow \infty} |P_n(z)|^{1/n} = |\Omega(z)|$ and (1.3) we obtain $\lim_{n \rightarrow \infty} \sup |a_n P_n(z_0)|^{1/n} > 1$, showing that the series $\sum_{n=0}^{\infty} a_n P_n(z_0)$ diverges.

Conversely, if (1.3) holds, then as above, we can show that the series $\sum_{n=0}^{\infty} a_n P_n(z)$ converges uniformly on compact subsets of D_R to a regular function, $f(z)$, say. If $f(z)$ had no singular point on L_R then it would be possible to extend $f(z)$ analytically to a bigger domain D_{R_0} , say. But then the first part of the theorem would give that

$$\lim_{n \rightarrow \infty} \sup |a_n|^{1/n} \leq 1/R_0 < 1/R, \text{ a contradiction. Hence the proof is completed.} \quad \square$$

In view of above theorem, there exists a sequence of polynomials converging uniformly on compact subsets of D_R to $f(z)$, it follows that this sequence converges in the norm of $L_p(D)$ also to f . If p_{n-1} denotes the collection of all polynomials of degree not exceeding $n - 1$ and we set

$$E_n^p(f) = \inf_{g \in p_{n-1}} \|f - g\|_{D,p}, \quad n = 1, 2, \dots,$$

then it is clear that $E_n^p(f)$ is a non increasing sequence tending to zero as $n \rightarrow \infty$. Our next theorem holds for $1 \leq p < \infty$.

Theorem 1.2. *If $f \in H(\overline{D}; R)$, then*

$$\limsup_{n \rightarrow \infty} [E_n^p(f)]^{1/n} = \frac{d}{R}. \quad (1.5)$$

Proof. If $f \in H(\overline{D}; R)$ we have, by (1.5)

$$\begin{aligned} E_n^p(f) &= \inf_{g \in p_{n-1}} \left(\int \int_D |f(z) - g(z)|^p dx dy \right)^{1/p} \leq \left(\int \int_D |f(z) - Q_{n-1}(z)|^p dx dy \right)^{1/p} \\ &\leq A^{1/p} \max_{z \in \overline{D}} |f(z) - Q_{n-1}(z)|, \end{aligned}$$

where $Q_{n-1}(z)$ is the polynomial of degree not exceeding $n - 1$ and A is the area of domain D . Using a result of (Markushevich, 1967), p. 114 for $d < r' < r < R$ and $n > n_0$ we get

$$E_n^p(f) \leq A^{1/p} \overline{M}(r, f) \left(\frac{r'}{r - r'} \right) (r'/r)^n \quad (1.6)$$

where $\overline{M}(r, f) = \max_{z \in L_r} |f(z)|$. This leads to $\lim_{n \rightarrow \infty} \sup [E_n^p(f)]^{1/n} \leq r'/r$.

Since the above relation holds for all r', r satisfying $d < r' < r < R$, we must have

$$\limsup_{n \rightarrow \infty} [E_n^p(f)]^{1/n} \leq \frac{d}{R}. \quad (1.7)$$

To obtain the reverse inequality in (1.7), we note that, since every $f \in H(\overline{D}; R)$ is in $H_2(D)$, there exists a closed orthonormal system $\{\chi_n(z)\}_{n=0}^\infty$ of polynomials in $H_2(D)$ such that f can be represented by its Fourier series with respect to the system $\{\chi_n(z)\}_{n=0}^\infty$ that converges uniformly on compact subsets of D_R to f . Thus

$$f(z) = \sum_{n=0}^{\infty} a_n \chi_n(z) \quad z \in D_R, \quad (1.8)$$

where $a_n = \int \int_D f(z) \overline{\chi_n(z)} dx dy$.

If $g \in p_{n-1}$, then

$$|a_n| = \left| \int \int_D (f(z) - g(z)) \overline{\chi_n(z)} dx dy \right| \leq \left(\int \int_D |f(z) - g(z)|^p dx dy \right)^{1/p} \left(\int \int_D |\overline{\chi_n(z)}|^{p/p-1} dx dy \right)^{1-1/p}.$$

Using (1.3) and the fact that the above inequality holds for every $g \in p_{n-1}$, we get, for $r^* > d$, $|a_n| \leq E_n^p(f) \cdot \overline{M}(r, f) \left(\frac{r^*}{d} \right)^n A^{1-1/p}$, by (1.7) we get $\lim_{n \rightarrow \infty} \sup [E_n^p(f)]^{1/n} \geq \frac{d}{r^*} \lim_{n \rightarrow \infty} \sup |a_n|^{1/n} = \frac{d^2}{r^* R}$.

Since the above inequality is valid for every $r^* > d$, we must have

$$\limsup_{n \rightarrow \infty} [E_n^p(f)]^{1/n} \geq \frac{d}{R}. \quad (1.9)$$

Combining (1.7) and (1.9) we get (1.5). □

2. Fast Growth and Approximation Errors

We now obtain relations that indicate how the growth of an $f \in H(\overline{D}; R)$ depends on $E_n^p(f)$ and vice versa.

For function $f \in H(\overline{D}; R)$, set

$$\rho_R(q) = \limsup_{r \rightarrow R} \frac{\log^{[q]} \overline{M}(r, f)}{\log(Rr/(R-r))} \quad (2.1)$$

where $\log^{[0]} \overline{M}(r, f) = \overline{M}(r, f)$ and $\log^{[q]} \overline{M}(r, f) = \log(\log^{[q-1]} \overline{M}(r, f))$, $q = 1, 2, \dots$. To avoid the trivial cases we shall assume throughout that $\overline{M}(r, f) \rightarrow \infty$ as $r \rightarrow R$.

Definition 2.1. A function $f \in H(\overline{D}; R)$, is said to have the index q if $\rho_R(q) < \infty$ and $\rho_R(q-1) = \infty$, $q = 1, 2, \dots$. If q is the index of $f(z)$, then $\rho_R(q)$ is called the q -order of f .

Definition 2.2. A function $f \in H(\overline{D}; R)$ and having the index q is called to have lower q -order $\lambda_R(q)$ if

$$\lambda_R(q) = \liminf_{r \rightarrow R} \frac{\log^{[q]} \overline{M}(r, f)}{\log(Rr/(R-r))}, q = 1, 2, \dots \quad (2.2)$$

Definition 2.3. A function $f \in H(\overline{D}; R)$ and having the index q is said to be of regular q -growth if $\rho_R(q) = \lambda_R(q)$, $q = 1, 2, \dots$, $f(z)$ is said to be of irregular q -growth if $\rho_R(q) > \lambda_R(q)$, $q = 1, 2, \dots$.

In 1980 Andre Giroux ([Giroux, 1980](#)) obtained necessary and sufficient conditions, in terms of the rate of decrease of the approximation error $E_n^p(f)$, such that $f \in L_p(D)$, $2 \leq p \leq \infty$, has an analytic continuation as an entire function having finite growth parameters. In 1982 Kapoor and Nautiyal ([Kapoor & Nautiyal, 1982](#)) considered the approximation error $E_n^p(f)$ on a Caratheodory domain and had extended the results of Giroux for the case $1 \leq p < 2$. All these results do not give any specific information about the growth when function is not entire and for the functions having rapidly increasing maximum modulus such that order of function is infinite. Although, Kumar ([Kumar, 2004, 2007b,a, 2010, 2011, 2013](#)) and Kumar and Mathur ([Kumar & Amit, 2006](#)) obtained some results in this direction but our results are different from all those of above papers.

In this paper an attempt has been made to study the growth of $f \in H(\overline{D}; R)$ involving $E_n^p(f)$ for $1 \leq p < \infty$ when f is not entire and having $\overline{M}(r, f) \rightarrow \infty$ as $r \rightarrow R$. To obtain the results in general setting we shall assume that $f \in H(\overline{D}; R)$ is represented by the gap power series $f(z) = \sum_{n=0}^{\infty} a_n z^{\lambda_n}$ where $\{\lambda_n\}_{n=0}^{\infty}$ is strictly increasing sequence of integers and $a_n \neq 0$ for all n .

Theorem 2.1. If $f \in H(\overline{D}; R)$ having the index q and q -order $\rho_R(q)$, then

$$\rho_R(q) + A(q) = \limsup_{n \rightarrow \infty} \frac{\log^{[q-1]} \lambda_n}{\log \lambda_n - \log^+ \log^+ ((E_{\lambda_n}^p(f))(R/d)^{\lambda_n})}, q = 2, 3, \dots, \quad (2.3)$$

where, $A(q) = 1$ if $q = 2$ and $A(q) = 0$ if $q = 3, 4, \dots$

Proof. If $f \in H(\overline{D}; R)$ is of q -order $\rho_R(q)$, given $\varepsilon > 0$, there exists $r_0(\varepsilon)$ such that, for $r_0 < r < R$, we have

$$\log^{[q-1]} \overline{M}(r, f) < \left(\frac{Rr}{R-r} \right)^{\rho_R(q)+\varepsilon}.$$

Using inequality (1.6) for $n > n_0$ and $d < r' < r < R, r_0 < r'$, we obtain

$$\begin{aligned} \log E_{\lambda_n}^{(p)}(f)(R/d)^{\lambda_n} &< \frac{1}{p} \log A + \exp^{[q-2]} \left(\frac{Rr}{R-r} \right)^{\rho_R(q)+\varepsilon} + \log \frac{r'}{r-r'} + \lambda_n \log \frac{r'}{d} + \lambda_n \log \frac{R}{r} \\ &< \frac{1}{p} \log A + \exp^{[q-2]} \left(\frac{Rr}{R-r} \right)^{\rho_R(q)+\varepsilon} + \log \frac{r'}{r-r'} + \lambda_n \left(\frac{r'-d}{d} \right) + \lambda_n \left(\frac{R-r}{r} \right). \end{aligned} \quad (2.4)$$

Let us consider r such that

$$\begin{aligned} \frac{Rr}{R-r} &= \left(\log^{[q-2]} (\lambda_n R / \rho_R(q) + \varepsilon) \right)^{1/(\rho_R(q)+A(q)+\varepsilon)}, \text{ and} \\ r' &= \lambda d + (1-\lambda)(Rd/r), 0 < \lambda < 1. \end{aligned} \quad (2.5)$$

For $q = 2$ the inequality (2.4) with (2.5) gives for $n > n_1$,

$$\log^+ E_{\lambda_n}^p(f)(R/d)^{\lambda_n} < M(\lambda_n R)^{(\rho_R(2)+\varepsilon)/(\rho_R(2)+1+\varepsilon)}$$

where M is a constant. It gives

$$\limsup_{n \rightarrow \infty} \frac{\log^+ \log^+ E_{\lambda_n}^p(f)(R/d)^{\lambda_n}}{\log \lambda_n} \leq \frac{\rho_R(2)}{\rho_R(2)+1}. \quad (2.6)$$

For $q = 3, 4, \dots$, the inequality (2.4) gives for $n > n_1$, with (2.5) that

$$\log^+ E_{\lambda_n}^p(f)(R/d)^{\lambda_n} < \exp^{[q-2]} \left(\log^{[q-2]} (\lambda_n R / \rho_R(q) + \varepsilon) \right) [1 + 0(1)],$$

from which a simple calculation would yield

$$\rho_R(q) \geq \limsup_{n \rightarrow \infty} \frac{\log^{[q-1]} \lambda_n}{\log \lambda_n - \log^+ \log^+ E_{\lambda_n}^p(f)(R/d)^{\lambda_n}}. \quad (2.7)$$

To prove that reverse inequality, use (1.8) and (1) to get, for $z \in \overline{D}_r, d < r^* < r < R$,

$$\begin{aligned} |f(z)| &\leq \sum_{n=0}^{\infty} |a_n| |\chi_{\lambda_n}(z)| \\ &\leq MA^{1-1/p} \sum_{n=0}^{\infty} E_n^p(f) \left(\frac{r^*}{d} \right)^n |\chi_{\lambda_n}(z)|. \end{aligned}$$

It is known that for any $r^* > d$ there exists a constant M^l such that

$$|\chi_n(z)| \leq M'(r^*/d)^n, n = 0, 1, 2, \dots, z \in \overline{D}.$$

Now for $d < r^* < r < R$, we get

$$\overline{M}(r, f) \leq M^* M^l A^{1-1/p} \sum_{n=0}^{\infty} E_n^p(f)(R/d)^{\lambda_n} \left(\frac{r^{*2} r}{d^2 R} \right)^{\lambda_n}.$$

Taking $r^* = \sqrt[p]{\lambda + (1-\lambda)(R/r)}$, $0 < \lambda < 1$, the above inequality gives

$$\overline{M}(r, f) \leq BM \left(\frac{\lambda r + (1-\lambda)R}{R}, G \right), \quad (2.8)$$

where B is constant, $G(s) = \sum_{n=0}^{\infty} E_n^p(f)(R/d)^{\lambda_n} s^{\lambda_n}$ and $M(t, G) = \max_{|s|=t} |G(s)|$. It can be easily seen that $G(s)$ is analytic in $|s| < 1$. If the q -order of $G(s)$ in unit disc is $\rho_0(q)$ then

$$\rho_R(q) = \limsup_{r \rightarrow R} \frac{\log^{[q]} \overline{M}(r, f)}{\log(Rr/(R-r))} \leq \limsup_{r \rightarrow R} \frac{\log^{[q]} M((\lambda r + (1-\lambda)R)/R, G)}{\log(Rr/(R-r))} = \rho_0(q).$$

Applying Theorem 1 of (Kapoor & Gopal, 1979) for $\rho_0(q)$, we obtain

$$\rho_R(q) + A(q) \leq \limsup_{n \rightarrow \infty} \frac{\log^{[q-1]} \lambda_n}{\log \lambda_n - \log^+ \log^+ E_{\lambda_n}^p(f)(R/d)^{\lambda_n}} q = 2, 3, \dots, \quad (2.9)$$

combining (2.6), (2.7) and (2.9) we get (2.3) i.e., the proof of theorem is completed. \square

Theorem 2.2. Let $f \in H(\overline{D}; R)$ having the index- q , q -order $\rho_R(q)$ and lower q -order $\beta_R(q)$ ($0 \leq \beta_R(q) \leq \infty$), then for any increasing sequence $\{n_k\}$ of natural numbers,

$$\beta_R(q) + A(q) \geq \liminf_{k \rightarrow \infty} \frac{\log^{[q-1]} \lambda_{n_{k-1}}}{\log \lambda_{n_k} - \log^+ \log^+ E_{\lambda_{n_k}}^p(f)(R/d)^{\lambda_{n_k}}}. \quad (2.10)$$

Proof. Let the right hand side of (2.10) be δ . Without loss of generality we can assume $\delta > 0$. For any ε such that $0 < \varepsilon < \delta$, and for all $k > k_0 = k_0(\varepsilon)$, we get

$$\log^+ E_{\lambda_{n_k}}^p(f)(R/d)^{\lambda_{n_k}} > \lambda_{n_k} (\log^{[q-2]} \lambda_{n_{k-1}})^{-1/(\delta-\varepsilon)}.$$

Choosing a sequence r_k such that $r_k \leq r \leq r_{k+1}$, where $\frac{R-r_k}{r_k} = \frac{1}{e} (\log^{[q-2]} \lambda_{n_{k-1}})^{1/(\delta-\varepsilon)}$.

Using inequality (1.6) we obtain

$$\begin{aligned} \log \overline{M}(r, f) &\geq \log E_{\lambda_{n_k}}^p(f)(R/d)^{\lambda_{n_k}} - \frac{1}{p} \log A - \log \frac{r'}{r-r'} - \lambda_{n_k} \log \frac{r'}{d} - \lambda_{n_k} \log R/r \\ &\geq \log E_{\lambda_{n_k}}^p(f)(R/d)^{\lambda_{n_k}} - \frac{1}{p} \log A - \log \frac{r'}{r_k-r'} - \lambda_{n_k} \log \frac{r'}{d} - \lambda_{n_k} \log R/r_k \\ &> \lambda_{n_k} (\log^{[q-2]} \lambda_{n_{k-1}})^{-1/(\delta-\varepsilon)} - \lambda_{n_k} \left(\frac{R-r_k}{r_k} \right) = (1-1/e) \lambda_{n_k} (\log^{[q-2]} \lambda_{n_{k-1}})^{-1/(\delta-\varepsilon)} \\ &> (e-1) \left(\frac{R-r}{r} \right) \exp^{[q-2]} \left(e \left(\frac{R-r}{r} \right) \right)^{-(\delta-\varepsilon)}. \end{aligned}$$

Now after a simple calculation the above estimate yields

$$\beta_R(q) + A(q) \geq \delta. \quad (2.11)$$

Hence the proof is completed. \square

To prove our next theorem we need the following lemma.

Lemma 2.1. Let $f(z) = \sum_{n=0}^{\infty} a_n z^{\lambda_n}$ be analytic in unit disc having index q , q -order $\rho(q) > 0$ and lower q -order $\beta(q)$. Further, let $\varphi(n) \equiv |a_n/a_{n+1}|^{1/(\lambda_{n+1}-\lambda_n)}$ forms a non-decreasing function of n for $n > n_0$ and

$$\beta(q) + A(q) = \liminf_{r \rightarrow 1} \frac{\log^{[q-1]} \nu(r)}{-\log(1-r)}. \quad (2.12)$$

Then

$$\beta(q) + A(q) = \liminf_{n \rightarrow \infty} \frac{\log^{[q-1]} \lambda_n}{\log \lambda_n - \log^+ \log^+ |a_n|}$$

where for $|z| = r$, $\mu(r) = \max_{n>0} \{|a_n| r^{\lambda_n}\}$, $\nu(r) = \max\{\lambda_n : \mu(r) = |a_n| r^{\lambda_n}\}$, $0 < r < 1$.

Proof. The proof of this lemma follows on the lines of a result in (Kapoor, 1972), so we omit the details. \square

A function f , analytic in unit disc is said to be admissible if its lower q -order satisfies (2.12).

Theorem 2.3. Let $f \in H(\overline{D}; R)$ having the index- q , q -order $\rho_R(q)$ and lower q -order $\beta_R(q)$. Further let $\varphi(n) = |E_{\lambda_n}/E_{\lambda_{n+1}}|^{1/(\lambda_{n+1}-\lambda_n)}$ forms a nondecreasing function of n for $n > n_0$. Then

$$\beta_R(q) + A(q) \leq \liminf_{n \rightarrow \infty} \frac{\log^{[q-1]} \lambda_n}{\log \lambda_n - \log^+ \log^+ E_{\lambda_n}^p(f)(R/d)^{\lambda_n}}. \quad (2.13)$$

Proof. Using (2.8), we get

$$\beta_R(q) = \liminf_{r \rightarrow R} \frac{\log^{[q]} \overline{M}(r, f)}{\log(Rr/(R-r))} \leq \liminf_{r \rightarrow R} \frac{\log^{[q]} M(((\lambda r + (1-\lambda)R)/R))}{\log(Rr/(R-r))} = \beta_0(q).$$

\square

It can be easily seen that $G(s)$ satisfies the hypothesis of Lemma 4. Applying Lemma 2.1 for $\beta_0(q)$, it gives

$$\beta_R(q) + A(q) \leq \liminf_{n \rightarrow \infty} \frac{\log^{[q-1]} \lambda_n}{\log \lambda_n - \log^+ \log^+ E_{\lambda_n}^p(f)(R/d)^{\lambda_n}} q = 2, 3, \dots,$$

combining Theorem 2.2 and 2.3 we get the following theorem:

Theorem 2.4. Let $f \in H(\overline{D}; R)$ having the index- q , q -order $\rho_R(q)$ and lower q -order $\beta_R(q)$. Further, let $\varphi(n) = |a_n/a_{n+1}|^{1/(\lambda_{n+1}-\lambda_n)}$ forms a nondecreasing function of n for $n > n_0$. Then

$$\beta_R(q) + A(q) = \liminf_{n \rightarrow \infty} \frac{\log^{[q-1]} \lambda_{n-1}}{\log \lambda_n - \log^+ \log^+ E_{\lambda_n}^p(f)(R/d)^{\lambda_n}}. \quad (2.14)$$

Theorem 2.5. Let $f \in H(\overline{D}; R)$ having the index- q , q -order $\rho_R(q)$ and lower q -order $\beta_R(q)$. Then

$$\beta_R(q) + A(q) = \max_{\{n_k\}} \left[\liminf_{k \rightarrow \infty} \frac{\log^{[q-1]} \lambda_{n_{k-1}}}{\log \lambda_{n_k} - \log^+ \log^+ E_{\lambda_{n_k}}^p(f)(R/d)^{\lambda_{n_k}}} \right], \quad (2.15)$$

where maximum in (2.15) is taken overall increasing sequences $\{n_k\}$ of natural numbers.

Proof. Let $S(s) = \sum_{k=0}^{\infty} E_{\lambda_{n_k}}^p(f)(R/d)^{\lambda_{n_k}} s^{\lambda_{n_k}}, |s| < 1$, where $\{\lambda_{n_k}\}_{k=0}^{\infty}$ is the sequence of elements in the range set of $\nu(r)$. It can be easily seen that $G(s)$ and $S(s)$ have the same maximum term. Hence, the q -order and lower q -order of $S(s)$ are the same as those of $G(s)$. Thus, $S(s)$ is of lower q -order $\beta_R(q)$. Further, let $\xi(n_k) = \max\{r : \nu(r) = \lambda_{n_k}\}$. Then, $\xi(n_k) = \varphi(n_k)$, and consequently, $\varphi(n_k)$ is an increasing function of k . Therefore, $S(s)$ satisfies the hypothesis of Theorem 2.4 and so by (2.15) we get

$$\beta_R(q) + A(q) = \liminf_{k \rightarrow \infty} \frac{\log^{[q-1]} \lambda_{n_{k-1}}}{\log \lambda_{n_k} - \log^+ \log^+ E_{\lambda_{n_k}}^p(f)(R/d)^{\lambda_{n_k}}}. \quad (2.16)$$

But from Theorem 2.3, we get

$$\beta_R(q) + A(q) \geq \max_{\{n_k\}} \left[\liminf_{k \rightarrow \infty} \frac{\log^{[q-1]} \lambda_{n_{k-1}}}{\log \lambda_{n_k} - \log^+ \log^+ E_{\lambda_{n_k}}^p(f)(R/d)^{\lambda_{n_k}}} \right]. \quad (2.17)$$

Combining (2.16) and (2.17) we get (2.15). Hence the proof is complete. \square

References

- Giroux, A. (1980). Approximation of entire functions over bounded domains. *Journal of Approximation Theory* **28**(1), 45 – 53.
- Kapoor, G. P. (1972). On the lower order of functions analytic in the unit disc. *Math. Japon.* **17**(1), 45–54.
- Kapoor, G.P. and A. Nautiyal (1982). Approximation of entire functions over carathodory domains. *Bulletin of the Australian Mathematical Society* **25**, 221–229.
- Kapoor, G.P. and K. Gopal (1979). On the coefficients of functions analytic in the unit disc having fast rates of growth. *Annali di Matematica Pura ed Applicata* **121**(1), 337–349.
- Kumar, D. (2004). Coefficients characterization for functions analytic in the polydisc with fast growth. *Math. Sci. Res. J.* **8**(4), 128–136.
- Kumar, D. (2007a). Necessary conditions for L^p - convergence of Lagrange interpolation in finite disc. *International Journal of Pure and Applied Mathematics* **40**(2), 153–164.
- Kumar, D. (2007b). On approximation and interpolation errors of an analytic functions. *Fasc. Math.* **38**, 17–36.
- Kumar, D. (2010). On the fast growth of analytic functions by means of Lagrange polynomial approximation and interpolation in C^N . *Fasc. Math.* **13**, 85–99.
- Kumar, D. (2011). Growth and weighted polynomial approximation of analytic functions. *Transylvanian Journal of Mathematics and Mechanics* **3**(1), 23–30.
- Kumar, D. (2013). Slow growth and optimal approximation of pseudoanalytic functions on the disc. *International Journal of Analysis and Applications* **2**(1), 26–37.
- Kumar, D. and M. Amit (2006). On the growth of coefficients of analytic functions. *Math. Sci. Res. J.* **10**(11), 286–295.
- Markushevich, A. I. (1967). *Theory of Functions of a Complex Variable, Vol.III. Revised English Edition. (Translated and Edited by Richard A. Silverman.* Prentice-Hall, Englewood Cliffs, New Jersey.