AAS 12-624

A $\mathbb{K}$-VECTOR APPROACH TO SAMPLING, INTERPOLATION, AND APPROXIMATION

Daniele Mortari* **and Jonathan Rogers†**

The **k**-vector search technique is a method designed to perform extremely fast range searching of large databases at computational cost independent of the size of the database. **k**-vector search algorithms have historically found application in satellite star-tracker navigation systems which index very large star catalogues repeatedly in the process of attitude estimation. Recently, the **k**-vector search algorithm has been applied to numerous other problem areas including non-uniform random variate sampling, interpolation of 1-D or 2-D tables, nonlinear function inversion, and solution of systems of nonlinear equations. This paper presents algorithms in which the **k**-vector search technique is used to solve each of these problems in a computationally-efficient manner. In instances where these tasks must be performed repeatedly on a static (or nearly-static) data set, the proposed **k**-vector-based algorithms offer an extremely fast solution technique that outperforms standard methods.

Truth is much too complicated to allow anything but approximations (John von Newman, 1947).

INTRODUCTION

The **k**-vector search method is a range searching technique devised [1] within the general problem of spacecraft attitude determination for the *lost-in-space* case, when no estimate of the spacecraft attitude is available. In particular, the technique was developed to accelerate the Star-ID problem of wide field-of-view (FOV) star trackers. Using the **k**-vector technique, the Star-ID Searchless Algorithm was first proposed in [2]. The **k**-vector search method requires the construction of a vector of integers of length n, the **k**-vector, whose entries contain information to solve the searching problem. Use of the **k**-vector vastly reduces the search effort required, which constitutes the heaviest computational burden of Star-ID algorithms. Since its initial development, the **k**-vector technique has been applied in various Star-ID approaches: the SP-Search [3], multiple FOV star trackers [4, 5], and the LISA algorithm [6] for the StarNav I test experiment onboard STS-107.

One of the most important considerations when using the **k**-vector is that it is primarily suitable only for *static databases* (e.g., inter-star angles are time-invariant). In other words, ideal performance is achieved if data is not inserted, deleted, or changed within the database. Recognizing that the **k**-vector technique is a general mathematical tool to solve the range searching problem, the authors extended the **k**-vector algorithm in [7] to explore its potential in new applications other than

*Professor, 746C H.R. Bright Bldg, Aerospace Engineering, Texas A&M University, College Station, TX 77843-3141, Tel.: (979) 845-0734, Fax: (979) 845-6051, AIAA Associate Fellow. IEEE senior member. E-mail: MORTARI@TAMU.EDU

†Assistant Professor, 746B H.R. Bright Bldg, Aerospace Engineering, Texas A&M University, College Station, TX 77843-3141, Tel.: (979) 862-3413, Fax: (979) 845-6051. E-mail: JROGERS@AERO.TAMU.EDU

the Star-ID problem. This new searching technique became the heart of "Pyramid" [8], the current state-of-the-art in Star-ID algorithms, which has been flying onboard the MIT HETE and HETE-2 missions as well as several other satellites. However, application of the **k**-vector technique to this point has largely been limited to the original Star-ID application.

This study explores techniques to apply **k**-vector search methods to a broader class of problems, of which Star-ID is one specific single application. Specific areas of interest include efficient nonlinear function inversion, minimal search table interpolation, and random variate sampling from an arbitrary density function. In each application, emphasis is placed on minimizing computational burden with respect to current state-of-the-art methods. The paper is organized as follows. First, background on the general **k**-vector search algorithm is presented. Then, a **k**-vector-based method to invert nonlinear functions at discrete values is outlined, and specific 1-D and 2-D examples are offered. The method is also highly effective in computing intersections of nonlinear functions of arbitrary dimension. Interpolation techniques using the **k**-vector are then discussed in the context of static aerodynamic databases, which are typically indexed repeatedly during atmospheric vehicle simulation. Finally, a **k**-vector approach to random variate sampling is presented with potential applications in the area of sensor simulation and particle filter resampling. Overall, the methods developed here promise to substantially reduce runtime requirements for many basic mathematical algorithms of interest to the science and engineering community.

Background on the k-vector

The range searching problem consists of identifying, within a large n-valued database $\mathbf{y}$ ($n \gg 1$), the set of all data elements $\mathbf{y}(\mathbf{I})$, where $\mathbf{I}$ is a vector of indices, such that $\mathbf{y}(\mathbf{I}) \in [y_a,\ y_b]$, where $y_a < y_b$. This problem is usually solved using the *Binary Search Technique* (BST), whose complexity is $\mathcal{O}(2\log_2 n)$ in sorted databases (where $2\log_2 n$ represents the number of comparisons in the worst case). Many variants of BSTs have been introduced [9] for use in particular applications in order to minimize the searching time.

The **k**-vector technique is summarized as follows. Let $\mathbf{y}$ be an data vector of n elements ($n \gg 1$) and $\mathbf{s}$ the same vector but sorted in ascending mode, i.e. $\mathbf{s}(i) \le \mathbf{s}(i+1)$. Let $\mathbf{s}(i) = \mathbf{y}(\mathbf{I}(i))$, where $\mathbf{I}$ is the integer vector of the sorting with length n. In particular, we have $y_{\min} = \min_i \mathbf{y}(i) = \mathbf{s}(1)$ and $y_{\max} = \max_i \mathbf{y}(i) = \mathbf{s}(n)$.

As shown in Fig. 1,* the line connecting the two extreme points, $[1, y_{\min}]$ and $[n, y_{\max}]$, has on average, $E_0 = n/(n-1)$ elements for each $d = (y_{\max} - y_{\min})/(n-1)$ step. However, we consider a slightly steeper line connecting the points $[1, y_{\min} - \delta\varepsilon]$ and $[n, y_{\max} + \delta\varepsilon]$, where $\delta\varepsilon = (n-1)\varepsilon$ and ε is the relative machine precision ($\varepsilon \approx 2.22 \times 10^{-16}$ for double precision arithmetic)†. This slightly steeper line assures $\mathbf{k}(1) = 0$ and $\mathbf{k}(n) = n$, thus simplifying the code by avoiding many index checks. The equation of this line can be written as

$$z(x) = m\,x + q \qquad \text{where} \qquad \begin{cases} m = (y_{\max} - y_{\min} + 2\delta\varepsilon)/(n-1) \\ q = y_{\min} - m - \delta\varepsilon \end{cases} \tag{1}$$

Setting $\mathbf{k}(1) = 0$ and $\mathbf{k}(n) = n$, the i-th element of the **k**-vector is

$$\mathbf{k}(i) = j \quad \text{where} \quad j \quad \text{is the greatest index such that} \quad \mathbf{s}(j) \le \mathbf{y}(\mathbf{I}(i)) \quad \text{is satisfied.} \tag{2}$$

*Figures 1 and 2 has been taken from Ref. [7].

†The expression provided for $\delta\varepsilon$ comes from the proportionality, $\dfrac{y_{\max} - y_{\min}}{\delta\varepsilon} = \dfrac{(y_{\max} - y_{\min})/(n-1)}{\varepsilon}$.

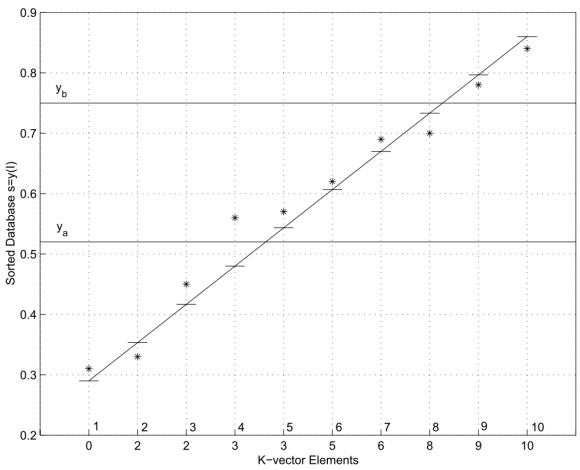

Figure 1. Example of k-vector Construction

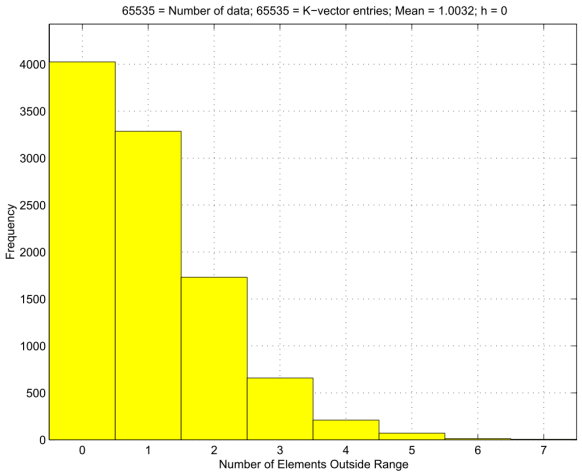

Figure 2. Histogram of E_0

From a practical point of view the value of $\mathbf{k}(i)$ represents the number of elements of the database vector $\mathbf{y}$ below the value $z(i)$, which is trivial to prove. As an example, Fig. 1 shows the construction of the $\mathbf{k}$-vector for a 10-element database. The elements of the database are shown by markers, the small horizontal lines are the $z(i)$ values, and the resulting $\mathbf{k}$-vector is given by $\mathbf{k} = \{0, 2, 2, 3, 3, 5, 6, 8, 9, 10\}^{\mathrm{T}}$.

Once the $\mathbf{k}$-vector has been built, the evaluation of the two indices identifying, in the $\mathbf{s}$ vector, all of the possible data falling within the range $[y_a,\ y_b]$, becomes a straightforward task that can be performed very quickly. In fact, the indices associated with these values in the $\mathbf{s}$ vector are simply provided as

$$j_b = \left\lfloor \frac{y_a - q}{m} \right\rfloor \qquad \text{and} \qquad j_t = \left\lceil \frac{y_b - q}{m} \right\rceil \tag{3}$$

where the function $\lfloor x \rfloor$ is the integer number immediately below x, and $\lceil x \rceil$ is the larger integer number next to x. In the example of Fig. 1, we have $j_b = 4$ and $j_t = 9$. Once the indices j_b and j_t are evaluated, it is possible to compute the range searching indices according to

$$k_{\text{start}} = \mathbf{k}(j_b) + 1 \qquad \text{and} \qquad k_{\text{end}} = \mathbf{k}(j_t) \tag{4}$$

Finally, the searched elements, $\mathbf{y}(i) \in [y_a,\ y_b]$, are the elements $\mathbf{y}(\mathbf{I}(k))$ where $k \in [k_{\text{start}}, k_{\text{end}}]$. In our example, however, the searched elements should be those identified by the range indices $k_{\text{start}} = 4$ and $k_{\text{end}} = 8$, while the proposed technique outputs $k_{\text{start}} = 4$ and $k_{\text{end}} = 9$. In the selected elements there is an additional extraneous element originating from the fact that, on average, the k_{start} and k_{end} elements each have 50% probability of not belonging to the $[y_a,\ z(j_a + 1)]$ and to the $[z(j_b),\ y_b]$ ranges, respectively. Therefore, if E_0 is the expected number of elements in the $[z(j),\ z(j + 1)]$ range, then the expected number of elements extraneous to the range $[y_a,\ y_b]$ is $2E_0/2 = E_0 = n/(n - 1) \approx 1$ for $n \gg 1$.

If the presence of these $E_0 \approx 1$ extraneous elements cannot be tolerated, two local searches are required to eliminate them at the extremes. These local searches are performed as follows:

$$\begin{cases} \text{check if} \quad \mathbf{y}(\mathbf{I}(k)) < y_a \quad \text{where} \quad k = k_{\text{start}}, \cdots \quad \text{(with positive step)} \\ \text{check if} \quad \mathbf{y}(\mathbf{I}(k)) > y_b \quad \text{where} \quad k = k_{\text{end}}, \cdots \quad \text{(with negative step)} \end{cases} \tag{5}$$

For large databases ($n \gg 1$), since $\lim_{n\to\infty} E_0 = 1$, the number of searched data can be approximated by $n_d = k_{\text{end}} - k_{\text{start}}$, because a number of $E_0 \approx 1$ extraneous elements are expected. Thus, *on average*, the method provides the solution with only three comparisons[‡], with complexity $\mathcal{O}(3)$ which is *independent from the database size* and whose value is much less than $\mathcal{O}(2\log_2 n)$ as required by BST. This value of complexity holds if the **k**-vector behavior is linear or quasi-linear with respect to the indices. Strong nonlinear behavior has been analyzed in Ref. [7]. For instance, a worst case scenario occurs when $(n-2)$ data fall within a single range step $[z(i), z(i+1)]$. In this case, however, the **k**-vector can be built within this range using $(n-2)$ data instead of n. Another worst case scenario occurs when the **k**-vector distribution is piecewise linear for which a different **k**-vector should be built for each different linear segment.

To validate the statistics of the number of extraneus elements, an 65535-element[§] vector y was been randomly created and the associated **k**-vector generated. Figure 2 shows the histogram of E_0 (the extraneous elements), obtained in 10,000 trials. The numerical mean value obtained is 1.0032, which agrees with the theoretical expected value $E_0 = 65535/(65535-1) = 1.000015$[¶].

K-VECTOR TO INVERT NONLINEAR N-DIMENSIONAL FUNCTIONS

The k-vector exhibits computational advantages when applied to static databases, and can thus be used effectively to invert nonlinear n-dimensional functions. The **k**-vector approach to perform function inversion is given as follows. Let $y = f(\mathbf{x})$ be a nonlinear function of n independent variables (where n is the dimensionality of $\mathbf{x}$). Examples of $n = 1$ mathematical problems involving the inversion of nonlinear functions are the following:

1. Find the values of x at which the function $y = f(x)$ assumes an assigned known value y. As elementary example is to find the roots of the nonlinear function $f(x) = 0$;

2. Find the values of a parameter p such that $\displaystyle\int_{x_{\min}}^{x_{\max}} f(x,p)\ \mathrm{d}x = y$, where $x_{\min}$ and $x_{\max}$ are assigned. For instance, this problem occurs when trying to invert the complete elliptic integral of the first kind (find k)

$$K(k) = \int_0^{\pi/2} \frac{1}{\sqrt{1 - k^2\,\sin^2\theta}}\ \mathrm{d}\theta = \int_0^1 \frac{1}{\sqrt{(1-t^2)(1-k^2\,t^2}}\ \mathrm{d}t$$

   or the complete elliptic integral of the second kind

$$E(k) = \int_0^{\pi/2} \sqrt{1 - k^2\,\sin^2\theta}\ \mathrm{d}\theta = \int_0^1 \frac{\sqrt{1-k^2\,t^2}}{\sqrt{1-t^2}}\ \mathrm{d}t$$

3. Find the value(s) of the integration bounds of an integral, $\displaystyle\int_{x_{\min}}^{x} f(\xi)\ \mathrm{d}\xi = y$ or $\displaystyle\int_{x}^{x_{\max}} f(\xi)\ \mathrm{d}\xi = y$. Some examples commonly found in scientific problems are:

   (a) the logarithmic integral function, $\mathrm{li}(x) = \displaystyle\int_0^x \frac{1}{\ln t}\ \mathrm{d}t$,

[‡]One for each check given in Eq. (5) plus the comparison needed to identify the $E_0 \approx 1$ extraneous element.

[§]The number of elements is $65535 = 2^{16} - 1$ so that each element index can be stored in 2-bytes.

[¶]The small difference between experimental and theoretical values comes from the fact that, when $k_{\text{end}} = k_{\text{start}} + 1$ occurs, then the number of expected extraneous elements becomes $E_0/2$.

(b) the exponential integral[‖], $\mathrm{Ei}(x) = \displaystyle\int_{-\infty}^{x} \frac{e^t}{t}\,\mathrm{d}t$,

(c) the sin and cosine integrals $\mathrm{Si}(x) = \displaystyle\int_0^x \frac{\sin t}{t}\,\mathrm{d}t$ and $\mathrm{Ci}(x) = \gamma + \ln x + \displaystyle\int_0^x \frac{\cos t - 1}{t}\,\mathrm{d}t$, where $\gamma = \displaystyle\lim_{n\to\infty}\left[\sum_{k=1}^{n}\frac{1}{k} - \ln(n)\right] = \lim_{b\to\infty}\int_1^b \left(\frac{1}{\lfloor x\rfloor} - \frac{1}{x}\right)\,\mathrm{d}x \approx 0.577215665$ is the Euler-Mascheroni constant

(d) the hyperbolic sine and cosine integrals, $\mathrm{Shi}(x) = \displaystyle\int_0^x \frac{\sinh t}{t}\,\mathrm{d}t$ and $\mathrm{Chi}(x) = \gamma + \ln x + \displaystyle\int_0^x \frac{\cosh t - 1}{t}\,\mathrm{d}t$

(e) the error function, $\mathrm{erf}(x) = \dfrac{2}{\sqrt{\pi}}\displaystyle\int_0^x e^{-t^2}\,\mathrm{d}t$,

(f) the Fresnel integrals, $S(x) = \displaystyle\int_0^x \sin(t^2)\,\mathrm{d}t$ and $C(x) = \displaystyle\int_0^x \cos(t^2)\,\mathrm{d}t$,

(g) the Dawson integral, $D_+(x) = e^{-x^2}\displaystyle\int_0^x e^{t^2}\,\mathrm{d}t$ and $D_-(x) = e^{x^2}\displaystyle\int_0^x e^{-t^2}\,\mathrm{d}t$,

(h) the Digamma function, $\psi(x) = \displaystyle\int_0^\infty \left(\frac{e^{-t}}{t} - \frac{e^{-xt}}{1-e^{-t}}\right)\,\mathrm{d}t$,

(i) the Polygamma function, $\psi^{(m)}(x) = (-1)^{m+1}\displaystyle\int_0^\infty \frac{t^m\,e^{-xt}}{1-e^{-t}}\,\mathrm{d}t$,

(j) and Bessel's integrals, $J_n(x) = \dfrac{1}{\pi}\displaystyle\int_0^\pi \cos(nt - x\sin t)\,\mathrm{d}t$.

The Methodology

The k-vector methodology for function inversion may be explained through an example. Consider the problem of inverting the Fresnel integral, $y = \displaystyle\int_0^x \sin(t^2)\,\mathrm{d}t$ in the domain $x \in [0, 5]$. The goal is to construct a numerical procedure that solves for *all* the roots, $x_k \in [0, 5]$, for any given value of the integral. Before proceeding with the description of the algorithm it is important to define the limits in applying the proposed methodology:

1. the proposed method is *not* suitable to invert a nonlinear function just for a single time, for which other methods may be more efficient. Since the proposed method requires some preprocessing efforts (see Fig. 4 for the algorithm), its efficiency is exposed when the function inversion problem must be performed *extensively* in some assigned range for the same nonlinear function. Once the preprocessing has been completed, the inverting function can be created and used anytime, either extensively or just for a single shot. For instance, the proposed methodology can be seen suitable to create fast built-in inverting functions for MATLAB.

[‖]To have an idea of the many applications of just $\mathrm{Ei}(x)$, Ref. [10] mentioned: a) Time-dependent heat transfer, b) Nonequilibrium groundwater flow in the Theis solution (called a well function), c) Radiative transfer in stellar atmospheres, d) Radial Diffusivity Equation for transient or unsteady state flow with line sources and sinks, and e) Solutions to the neutron transport equation in simplified 1-D geometries.

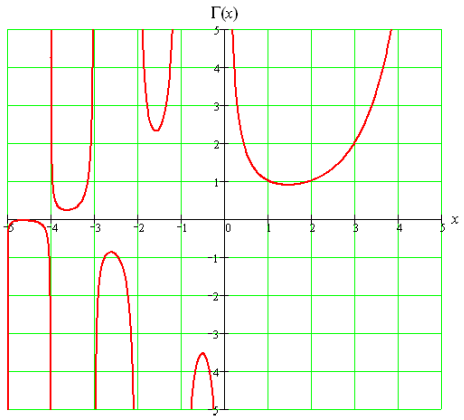

Figure 3. Example with singularities: the Gamma function $\Gamma(x)$

2. the proposed method is not suggested to invert functions in ranges where singularities occur, $f(x) = \pm\infty$, such as for the the Gamma function, $\Gamma(x) = \int_0^\infty e^{-t}t^{x-1}\ \mathrm{d}t$, (see Fig. 3). In general, function discontinuities are tolerated, but not singularities. When singularities are present in the range of interest the solution lies in either a) building piecewise inverting functions for all the ranges between subsequent singularities or b) built the inverting function not for the whole $[x_{\min}, x_{\max}]$ range, but for a limited $y \in [y_{\min}, y_{\max}]$ range.

The flowchart given in Fig. 4 shows that the method proposed to invert nonlinear functions is comprised of two separate tasks. The first is a preprocessing step which is only performed once. The second portion of the algorithm (right side of Fig. 4) outlines the technique to actually perform the function inversion. Input data to the algorithm are: 1) the nonlinear function $y = f(x)$, the range of interest, $[x_{\min}, x_{\max}]$, and the number of points, n, to discretize $f(x)$. Algorithm outputs are: 1) the number of roots, N_{root}, and the vector containing the roots X_e for any input value y (comprised of N_{root} elements).

The same technique can be used to invert 2-dimensional functions such as the Beta function, $B(x,y) = \int_0^1 t^{x-1}(1-t)^{y-1}\ \mathrm{d}t$, the upper and lower incomplete gamma functions, $\Gamma(x,y) = \int_y^\infty t^{x-1}e^{-t}\ \mathrm{d}t$ and $\gamma(x,y) = \int_0^y t^{x-1}e^{-t}\ \mathrm{d}t$, as well as those given in Table 1.

Table 1. Examples of multi-minima 2-dimensional functions

Shubert's function:	$z = -\sum_{i=1}^{5}\sum_{j=1}^{5}\{i\ \cos[(i+1)x+1]\}\{j\cos[(j+1)y+1]\}$
Six-hump camel back function:	$z = 400[(4 - 2.1x^2/4 + x^4/8)x^2/4 + xy/4 + (y^2/4 - 1)y^2]/6$

Figure 5 and 6 show the application of the **k**-vector to invert the Shubert's and the Six-hump camel back functions, respectively.

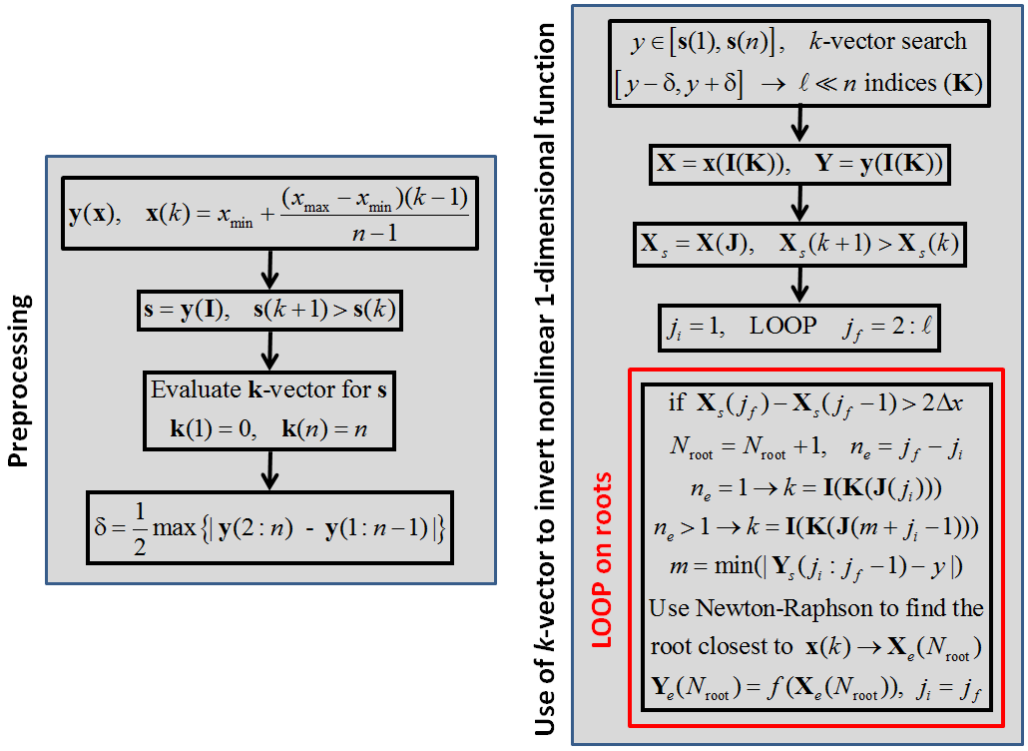

Figure 4. k-vector approach to invert the nonlinear $y = f(x)$ function in $x \in [x_{\min}, x_{\max}]$ using $n \gg 1$ points.

Solving Systems of two Nonlinear n-Dimensional Functions

The proposed methodology can be applied in numerous interesting applications, including the problem of finding solutions (values of $\mathbf{x}$) of the following set of two equations:

$$y = f(\mathbf{x}) \qquad \text{and} \qquad y = g(\mathbf{x}) + c \tag{6}$$

where $f(\mathbf{x})$ and $g(\mathbf{x})$ can be any two given nonlinear functions and c a constant variable whose value may change. It is straightforward to show that Eq. (6) is equivalent to solving the following equation

$$c = f(\mathbf{x}) - g(\mathbf{x}) = h(\mathbf{x}). \tag{7}$$

which can be done using the approach described in the previous section.

A 1-dimensional examples of this problem is shown in Fig. 7 for the Dawson function $f(x) = D_+(x) = e^{-x^2}\int_0^x e^{t^2}\,\mathrm{d}t$ and Sinc function $g(x) = \dfrac{\sin x}{x}$, while Fig. 8 shows a 2-dimensional example using the Shubert's and the Six-hump camel back functions given in Table 1.

Evaluation of iso-surfaces

Many scientific problems (i.e., computational fluid dynamics (CFD), combustion, atmospheric sciences, etc.) provide the 3-dimensional distribution of some physical quantity such as temperature, pressure, velocity, etc. It is of great interest to be able to quickly identify the surface associated with

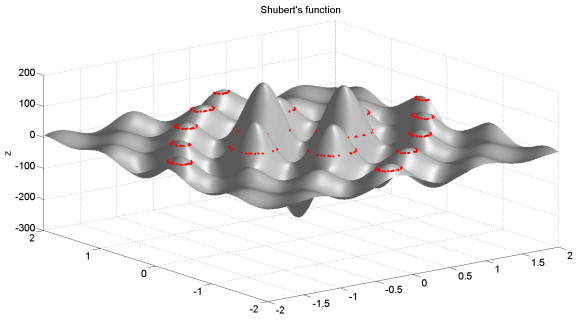

Figure 5. Inverting the Shubert's function

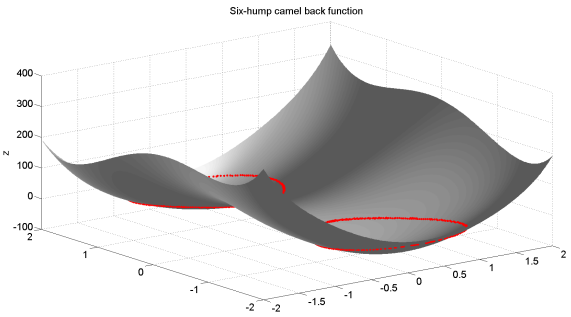

Figure 6. Inverting the Six-hump camel back function

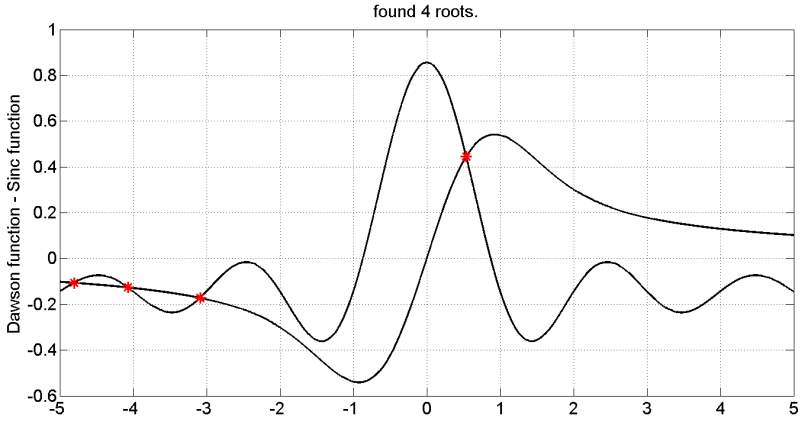

Figure 7. Solving for the intersection of two nonlinear 1-D functions

a given value of that physical quantity, typically called the iso-surface. One example in CFD is to find the stagnation surface (zero velocity) for an assigned distribution of velocity. Finding these iso-surfaces is usually a tedious and complicated problem as points on the surface are identified by gradient-based search algorithms or similar methods. However, the **k**-vector can solve this problem almost instantaneously, once the required preprocessing to build the **k**-vector has been completed.

Consider the problem of finding the iso-surface associated with temperature T_a in a given temperature distribution, $\mathbf{T}(x_k, y_k, z_k),\ k \in [1, N]$. The 3-dimensional matrix $\mathbf{T}$ is flattened into a vector, which is then sorted in ascending order. Then the **k**-vector is built for the sorted vector, and finally the iso-surface is identified by performing a **k**-vector range search between the values $T_a - \delta T$ and $T_a + \delta T$, where δT is a function of the total number of points and the maximum difference between contiguous elements. The importance of this type of problem will motivate future investigations dedicated to optimizing iso-surface identification using parallel computing.

Approximate Solution(s) of Nonlinear Diophantine Equations

The **k**-vector can also be used to rapidly approximate solutions to Diophantine Equations. Again, an example is selected to demonstrate the methodology. Consider the following nonlinear Diophantine equation

$$y = \frac{\pi\, z_1\, z_2 - z_3}{z_1 + e\, z_2\, z_3} = f(z_1, z_2, z_3) \tag{8}$$

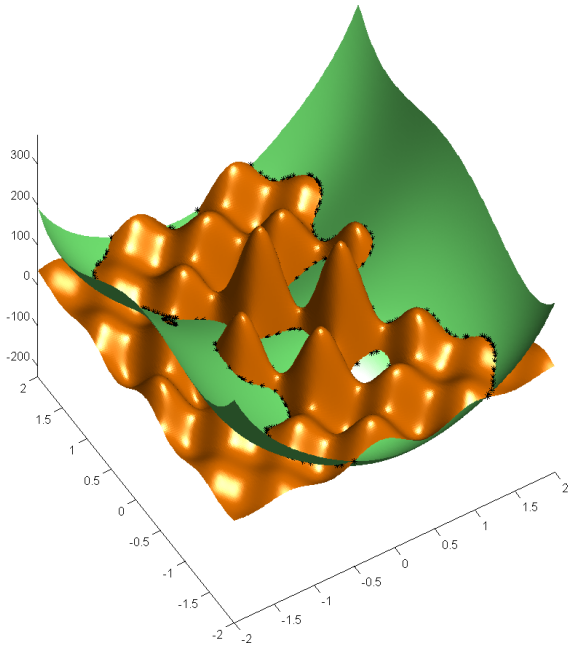

Figure 8. Solving for the intersection of two nonlinear 2-D functions

where the nonlinear Diophantine equation f may be general. In the selected specific example $e \approx 2.71828$ is Euler's number and $z_1, z_2, z_3 \in \mathbb{Z}^{(1)}$ are unknown integers. For a random real value of y, there may be no combination of z_1, z_2, and z_3 perfectly satisfying Eq. (8). However, the problem we would like to solve is: identify the best combinations of z_1, z_2, and z_3 (within given feasible ranges) such that $|y - f(z_1, z_2, z_3)|$ is minimum for a given value of y. Again, the proposed method is not satisfactory for a single approximation, but becomes highly efficient when approximate solution(s) of the same f function must be found for several different values of y.

To explain the approach consider the "feasible" ranges $z_1, z_2, z_3 \in [1, 40]$. The algorithm proceeds as follows:

1. Evaluate the function $g = f(z_1, z_2, z_3)$ at all points ($40^3 = 64,000$ points);
2. Sort the vector g in ascending order, obtaining the vector s;
3. Evaluate the **k**-vector for the s database;
4. Set $\delta = \max(s_k - s_{k-1})/N$, where N is a positive number (e.g., $N = 5$);
5. Randomly select a value of y between $\min(f) = s(1) + \delta$ and $\max(f) = s(64,000) - \delta$;
   (a) Perform the **k**-vector range searching in the range $[y - \delta, y + \delta]$;
   (b) if no element is found in the desired range, set $\delta = 2\delta$ and go back one step, otherwise continue;
6. The elements found are the best combinations that minimize $|y - f(z_1, z_2, z_3)|$.

Using $N = 5$ and $y = 27.4668$ the combination, $z_1 = 32$, $z_2 = 34$, and $z_3 = 1$, giving $y = 27.4635$, is obtained after doubling δ 3 times.

INTERPOLATION

Another application of the **k**-vector lies in fast interpolation of static N-D lookup tables. A common example in which fast interpolation is required is dynamic vehicle simulation, in which model parameters may be functions of the current state. In this case, at each timestep the current state value is used to find the appropriate index in the parameter lookup tables, and linear or higher-order interpolation is carried out between neighboring points to solve for the exact parameter value. In practice, the table index from the previous timestep is often stored and used as the basis for the search at the current timestep; however, this is efficient only in cases where the index variable changes slowly over each timestep. For high-resolution tables, this is not always true and **k**-vector interpolation will prove to be more efficient.

Consider a 1-D lookup table with discrete x coordinates $\mathbf{x} = \{x_1, x_2, \cdots, x_n\}^{\mathrm{T}}$ mapped to y coordinates $\mathbf{y} = \{y_1, y_2, \cdots, y_n\}^{\mathrm{T}}$. **k**-vector interpolation is initiated first by constructing the **k**-vector offline (i.e., before the dynamic simulation begins). In this case, the **k**-vector is given by $\mathbf{k} = \{k_1, k_2, \cdots, k_n\}^{\mathrm{T}}$ where k_j is the number of x values below a line extending from points $[1, -\delta]$ to $[n, x_n + \delta]$ at the j-th index. At a given simulation timestep, let the current value of x be denoted x^*. Then the **k**-vector index is computed according to

$$j = \text{floor}\left(\frac{x^* - \delta - q}{m}\right) \tag{9}$$

where $m = (x_n - x_1 + 2\delta)/(n-1)$. Then, to compute the interpolated value y^*, linear (or higher-order) interpolation is performed between x_{k_j} and x_{k_j+1}.

Aerodynamic Coefficient Look-up Table

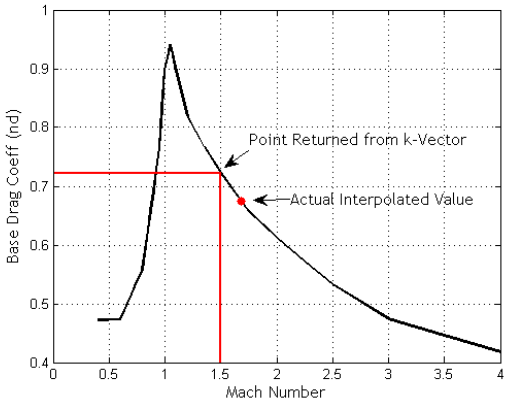

Figure 9. Drag Coefficient vs Mach Number.

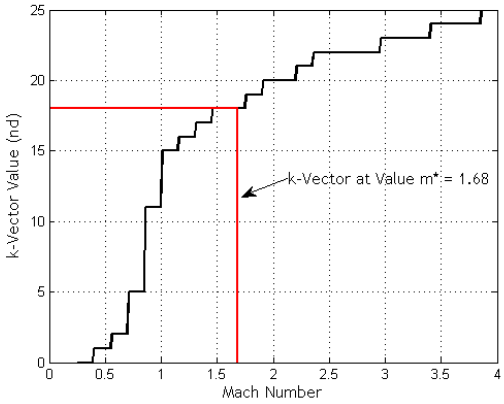

Figure 10. k-vector for Lookup Table.

Interpolation is an integral component of atmospheric vehicle dynamic simulation. Specifically, static databases loaded in memory hold aerodynamic parameters that are typically dependent on Mach-number [19] and other parameters. At each simulation timestep, the database must be searched for correct aerodynamic coefficients at the current Mach number, which involves searching for data near the current Mach number followed by interpolation between the nearest Mach values in the lookup table. K-Vector searching can locate the nearest Mach values and greatly increase computational efficiency since there is essentially no searching required.

Consider the zero-yaw drag force coefficient of the standard Army Navy Finner artillery projectile [20]. Figure 9 shows the Mach number dependence of this drag coefficient using 25 discrete points. Prior to a simulation, the $\mathbf{k}$-vector can be constructed strictly from the Mach number data as shown in Figure 10. This $\mathbf{k}$-vector represents the number of points whose corresponding Mach number is below the point on a line at the given index (see above for $\mathbf{k}$-vector construction algorithm). The static $\mathbf{k}$-vector is then stored in memory, as are the slope m and the intercept q used to build it. Suppose at the current timestep, the projectile is traveling at local Mach number 1.68. K-Vector interpolation of the drag coefficient table proceeds as follows. First, the $\mathbf{k}$-vector index is computed using Equation 9, yielding a value of 9. The $\mathbf{k}$-vector value at index 9 is 18, meaning that the 18th Mach point (corresponding to Mach 1.5) is the point below Mach 1.68, as shown in Figure 9. Thus linear interpolation can be immediately performed between the 18th and 19th Mach point with zero searching required. If model parameters are not reliably close to one another during successive timesteps, the performance of a few algebraic calculations using the $\mathbf{k}$-vector instead of searching methods could yield a significant reduction in runtime. This is particularly true of dynamic models that rely on large aerodynamic databases such as re-entry vehicles. Note that k-vector interpolation will only outperform standard search techniques (like BST) when databases are relatively large.

SAMPLING

$\mathbf{k}$-vector techniques can also be used for intensive random data generation within the range $x \in [x_{\min}, x_{\max}]$, for an arbitrary continuous (or discrete) distribution, $p(x)$. Let $\mathbf{p}(k)$ be a statistical realization of the density function $p(x)$ composed of $N \gg 1$ points, where $k \in [1, N]$. Also, let $\mathbf{s}(k)$ be the vector $\mathbf{p}(k)$ sorted in ascending order, i.e., $\mathbf{s} = \mathbf{p}(\mathbf{I})$, where $\mathbf{I}$ is the sorting index vector and $\mathbf{s}(k+1) \geq \mathbf{s}(k)$ is the ascending sorting condition. Let $\mathbf{k}$ be the $\mathbf{k}$-vector associated with the $\mathbf{s}$ vector. Using the sorted vector the range of the $\mathbf{p}$ vector is defined between $p_{\min} = \mathbf{s}(1)$ and $p_{\max} = \mathbf{s}(N)$.

Sampled data is then generated by performing the following steps:

1. Select $y \in [p_{\min}, p_{\max}]$ according to a uniform distribution.
2. Find the roots, $\mathbf{X}_e(k)$, which solve $y = p(x)$, using the $\mathbf{k}$-vector technique to invert the non-linear function $p(x)$. The number of roots will be an even number if the boundary conditions satisfy $p(x_{\min}) = p(x_{\max}) = 0$, otherwise $x_{\min}$ and/or $x_{\max}$ must be added as the "first" and/or "last" roots, respectively.
3. For each subsequent pair of roots, $\mathbf{X}_e(k)$ and $\mathbf{X}_e(k+1)$, exactly

$$n_k = \text{round}\left(N_\ell \frac{\mathbf{X}_e(k+1) - \mathbf{X}_e(k)}{x_{\max} - x_{\min}}\right) \tag{10}$$

   data points can be uniformly generated between $\mathbf{X}_e(k)$ and $\mathbf{X}_e(k+1)$.

Figure 11 shows an example function $p(x) = S(x)$, which the the Fresnel integral, and the histogram of 10,000,000 data points generated by this method where $N_\ell = 10,000$.

This method can be seen as a flipped *Rejection Sampling* [18] with *no rejections*. In fact, in rejection sampling, a sample is generated if, for a uniformly distributed value of $x \in [x_{\min}, x_{\max}]$, we obtain $p(x) \leq y$, where $y \in [p_{\min}, p_{\max}]$ is also uniformly distributed. In the $\mathbf{k}$-vector sampling method, for each random value of $y \in [p_{\min}, p_{\max}]$, Eq. (10) provides the number of data points

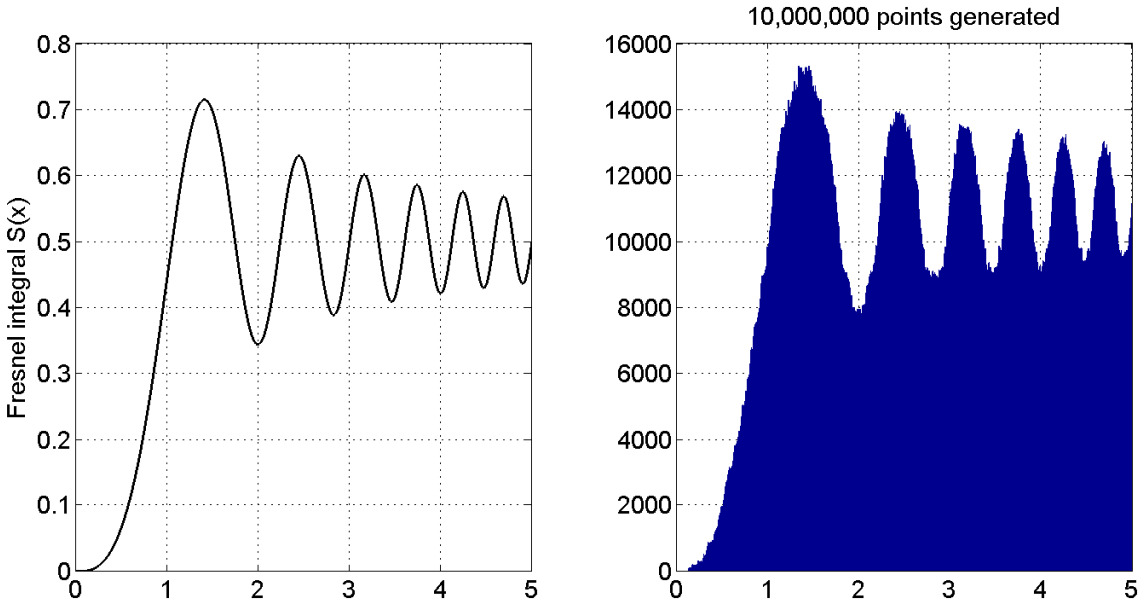

Figure 11. Sampling example. Distribution (left) and generated numbers histogram (right)

to be generated according to a uniform distribution within the range of neighboring roots, $\Delta x_k = \mathbf{X}_e(k+1) - \mathbf{X}_e(k)$. Equation (10) tells us that if the range between neighboring roots is the entire $[x_{\min}, x_{\max}]$ range, then N_ℓ data points can be uniformly generated in the same range.

Attitude data simulation using von Mises-Fisher distribution

As a particular example of a sampling application, consider simulation of line-of-sight measurements affected by noise which is described by the von Mises-Fisher distribution. In directional statistics, the von Mises-Fisher distribution [15, 16] is a probability distribution on the $(d-1)$-dimensional sphere in $\mathbb{R}^d$. The probability density function of the von Mises-Fisher distribution for the random d-dimensional unit-vector $\mathbf{b}$ is given by

$$f_d(\mathbf{b}, \bar{\mathbf{b}}, k) = C_d(k)\, e^{(k\,\bar{\mathbf{b}}^{\mathrm{T}}\mathbf{b})} \qquad \text{where} \qquad C_d(k) = \frac{k^{d/2-1}}{(2\pi)^{d/2}\, I_{d/2-1}(k)} \tag{11}$$

is the normalization constant, $k \geq 0$, $\|\bar{\mathbf{b}}\| = 1$, and I_v denotes the modified Bessel function of the first kind and order v.

In the most important case of $d = 3$, the von Mises-Fisher distribution represents the correct Gaussian distribution on the topological space of a sphere.** In this dimensional space the normalization constant reduces to

$$C_3(k) = \frac{k}{4\pi \sinh k} = \frac{k}{2\pi(e^k - e^{-k})} \tag{12}$$

The parameter k is called the *concentration parameter* and is the equivalent of the standard deviation for the Gaussian distribution. However, the greater the value of k, the higher the concentration

**The von Mises-Fisher distribution for $p = 3$, also called the Fisher distribution, was first used to model the interaction of dipoles in an electric field [17]. Other applications are found in geology, bioinformatics, and text mining.

of the distribution around $\bar{\mathbf{b}}$ (that is, the opposite concentration given by standard deviation). The distribution is unimodal for $k > 0$, and is uniform on the sphere for $k = 0$.

This distribution is important to aerospace engineering applications specifically in simulation of noise for line-of-sight attitude sensors. The direction $\bar{\mathbf{b}}$, called the mean direction, is the noise-free direction while the inner product, $\bar{\mathbf{b}}^{\mathrm{T}}\mathbf{b} = \cos\varepsilon$, quantifies the deviation of the measurement from the true value.

For an assigned value of k (specific for each sensor) the von Mises-Fisher distribution function (see Fig. 12 for $k = 10$) can be discretized and sampled data can be generated as described in the sampling section above.

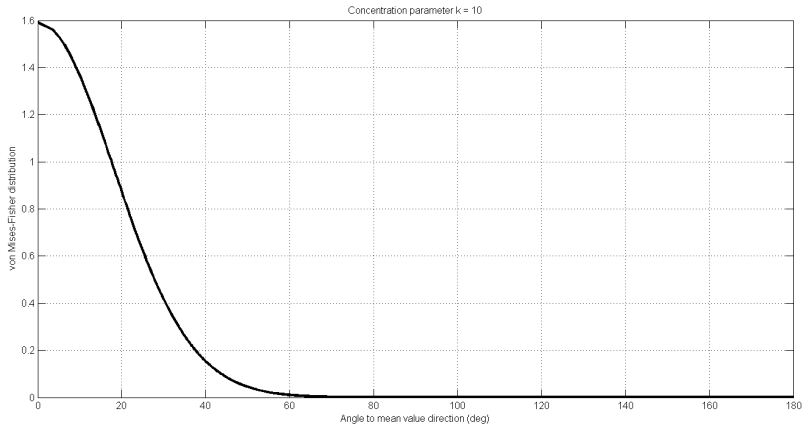

Figure 12. von Mises-Fisher distribution for $k = 10$

At each random value of $f_3(\cos\varepsilon, k)$, ranging from 0 to a maximum value of $C_3(k)\exp(k)$, n_k values of ε can be generated as uniformly generated between $\varepsilon = 0$ and the root $\varepsilon_{\mathrm{root}}$, which is computed by inverting the distribution function. The direction $\mathbf{b}$ is then obtained by: 1) randomly generated a unit vector $\hat{\mathbf{r}}$, 2) evaluating the direction (unit-vector) of $\hat{\mathbf{e}} \equiv \hat{\mathbf{r}} \times \bar{\mathbf{b}}$, 3) building the orthogonal matrix performing rigid rotation about $\hat{\mathbf{e}}$ of the angle ε, and 4) create the simulated direction (affected by noise) by rigid rotation

$$\mathbf{b} = R(\hat{\mathbf{e}}, \varepsilon)\,\bar{\mathbf{b}} \tag{13}$$

Particle Filter Resampling

Another potential application of k-vector sampling can be found in the field of nonlinear filtering. Sequential importance sampling filters, also called Bayesian bootstrap or particle filters, have recently become popular for a variety of real-time applications including target tracking and vehicle state estimation. At the same time, computational efficiency remains a serious problem and has restricted the use of these filters in many real-time implementations. The idea behind particle filtering is to represent the possibilities for the current state by a discrete set of N samples, or "particles". At a given measurement time, the set of measurements is mapped onto the *a priori* particle set to form a weight, or likelihood, associated with each particle. The posterior density function can be construction from the set of samples and associated weights. The particle set is resampled according to the posterior density. Finally, all particles are propagated forward to the next measurement time using a nonlinear dynamic model. The resampling step, in which a new particle set is constructed

based on the posterior density function, may be accelerated using efficient sampling techniques such as **k**-vector sampling.

To develop the details of **k**-vector resampling for particle filters, let the set of particles at the current timestep k be $\{\mathbf{x}_k^i\}_{i=1}^N$. Each particle has an associated normalized weight w_k^i such that $\sum_{i=1}^{N} w_k^i = 1$. The resampling step involves generating a new set of particles $\{\mathbf{x}_k^{i*}\}_{i=1}^N$ by sampling with replacement from a discrete approximation of the posterior density function given by

$$p(\mathbf{x}_k|\mathbf{z}_{1:k}) \approx \sum_{i=1}^{N} w_k^i \delta(\mathbf{x}_k - \mathbf{x}_k^i)$$

where $\mathbf{z}_{1:k}$ represents the measurements from the initial timestep through timestep k. Resampling of the approximate posterior distribution is performed so that the probability of a new particle $\mathbf{x}_k^{i*}$ being equal to the original particle $\mathbf{x}_k^j$ is w_k^j (i.e., $\text{Pr}(\mathbf{x}_k^{i*} = \mathbf{x}_k^j) = w_k^j$). The resulting particle set is in fact a realization of the discrete posterior density and thus after resampling all particle weights are equal.

Resampling algorithms typically require calculation of the cumulative likelihoods $Q_j = \sum_{i=1}^{j} w_k^i$ for $j = 1, \cdots, N$. The binary search resampling method is an $\mathcal{O}(N \log N)$ technique in which uniform random variates $(u_i)_{i=1,\cdots,N}$ are generated and binary search is used to find the value j, and hence $\mathbf{x}_k^j$, such that

$$Q_{j-1} < u_i < Q_j$$

where $Q_0 = 0$. However, this method is inefficient, and more efficient methods have been developed including stratified resampling [11], residual sampling [11], and a method based on order statistics [12]. These methods are of $\mathcal{O}(N)$ complexity and thus scale nicely with the particle set size. Systematic resampling [13] is another attractive $\mathcal{O}(N)$ resampling scheme which is relatively easy to implement.

A **k**-vector resampling technique which requires $\mathcal{O}(N)$ operations is outlined in Algorithm 1. The algorithm commences at the beginning of the resampling step, where N weights w_k^i correspond to N particles $\mathbf{x}_k^i$. This **k**-vector resampling algorithm was compared a $\mathcal{O}(N)$ binary search resampling scheme for several nonlinear problems, including a benchmark 1-D system used throughout the nonlinear filtering literature [13, 14]. Figure 13 shows a runtime comparison of the resampling step between these two methods for the benchmark 1-D system. Note that the k-vector sampling technique demonstrates approximately one order of magnitude runtime reduction for all particle set sizes.

Additional comparisons were carried out using a bearings-only target tracking example described in [12], in which BST, k-vector, systematic, and stratified sampling methods were compared. For all tests, a trend became evident that the computational requirements for **k**-vector and systematic resampling are roughly equivalent. In MATLAB implementations of the particle filter, the **k**-vector scheme outperformed systematic resampling (the fastest of the $\mathcal{O}(N)$ schemes) by approximately 3 times due to MATLAB's optimization of vector operations. When implemented in C/C++ however, the **k**-vector and systematic resampling schemes showed roughly equivalent performance. In fact, the particle filtering algorithm does not expose the true benefits of **k**-vector sampling, which lie

Algorithm 1 $\mathcal{O}(N)$ **k**-vector Particle Filter Resampling Algorithm

Compute slope and intercept of line used to build **k**-vector:
$m = (1 - w_k^1 + 2\delta)/(N-1)$
$q = w_k^1 - \delta - m$
CumInt = w_k^1
$j = 1$
for $i = 2 : N$ **do**
  $Y(i) = m * i + q$
  while CumInt ¡ $Y(i)$ **do**
    $j = j + 1$
    CumInt = CumInt + w_k^j
  $k(i) = j - 1$
Set $k(N) = N$
Generate N equally-spaced points between 0 and 1 $(y_j)_{j=1,\cdots,N}$
For each point y_j, output $\mathbf{x}_k^i$ where
$i = k(floor((w_k^1 + (1 - w_k^1)y_j - q)/m)) + 1$

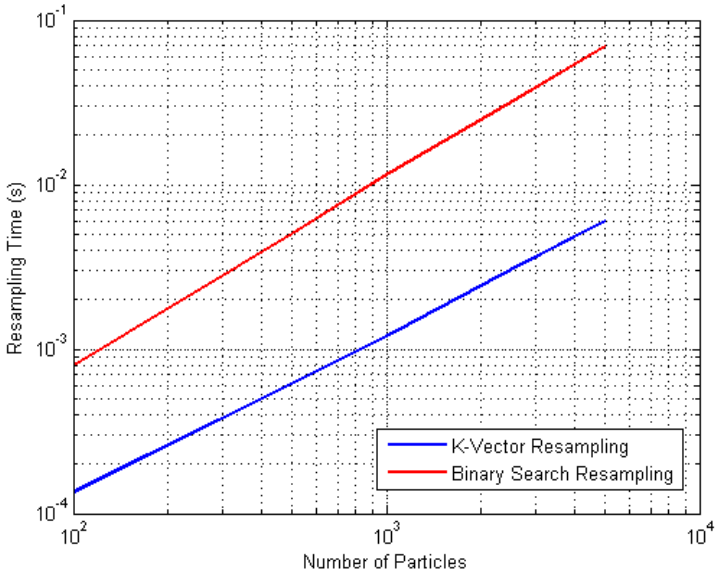

Figure 13. Particle Filter Runtime Comparison for Benchmark 1-D Nonlinear System [13, 14].

in the ability to sample efficiently from a static distribution repeatedly. In the case of a static distribution, the **k**-vector corresponding to the distribution can be saved between sampling instances and used again. However, the posterior pdf used during particle filtering resampling changes from timestep to timestep, and thus the **k**-vector must be recomputed every timestep which limits the overall efficiency of the sampling scheme.

CONCLUSIONS

Algorithms leveraging the **k**-vector search technique for random variate sampling, data interpolation, nonlinear function inversion, and solution of nonlinear systems were outlined. One highly attractive feature of the proposed **k**-vector algorithms applied to given datasets is that computational complexity is independent of the size of the dataset. Examples of one-dimensional and multi-dimensional nonlinear function inversion were provided, demonstrating that the proposed **k**-vector-based technique successfully finds all roots of a function simultaneously at essentially no extra computational cost. Once the **k**-vector is constructed for a discrete approximation to the function, the inversion procedure requires very little computation. Additional examples of a **k**-vector based sampling algorithm demonstrated the ability to generate large realizations of arbitrary probability densities, again at minimal computational cost. Finally, a particle filter resampling algorithm is proposed that exhibits similar complexity to the fastest resampling algorithms currently available. Overall, **k**-vector searching has proven a highly attractive technique for a wide variety of mathematical problems involving large data sets or high computational burden.

Acknowledgments

The authors would like to dedicate this work to Dr Jer-Nan Juang.

REFERENCES

[1] Mortari, D. "A Fast On-Board Autonomous Attitude Determination System based on a new Star-ID Technique for a Wide FOV Star Tracker," Advances in the Astronautical Sciences, Vol. 93, Pt. II, pp. 893-903.

[2] Mortari, D. "Search-Less Algorithm for Star Pattern Recognition," *The Journal of the Astronautical Sciences*, Vol. 45, No. 2, April-June 1997, pp. 179-194.

[3] Mortari, D. "SP-Search: A New Algorithm for Star Pattern Recognition," Advances in the Astronautical Sciences, Vol. 102, Pt. II, pp. 1165-1174.

[4] Mortari, D., Pollock, T.C., and Junkins, J.L. "Towards the Most Accurate Attitude Determination System Using Star Trackers," Advances in the Astronautical Sciences, Vol. 99, Pt. II, pp. 839-850.

[5] Mortari, D., and Angelucci, M. "Star Pattern Recognition and Mirror Assembly Misalignment for DIGISTAR II and III Star Sensors," Advances in the Astronautical Sciences, Vol. 102, Pt. II, pp. 1175-1184.

[6] Ju, G., Kim, Y.H., Pollock, C.T., Junkins, L.J., Juang, N.J., and Mortari, D. "Lost-In-Space: A Star Pattern Recognition and Attitude Estimation Approach for the Case of No A Priori Attitude Information," Paper AAS 00-004 of the 2000 AAS Guidance & Control Conference, Breckenridge, CO, Feb. 2-6, 2000.

[7] Mortari, D. and Neta, B. "**k**-vector Range Searching Techniques," 10th Annual AIAA/AAS Space Flight Mechanics Meeting. Paper AAS 00-128, Clearwater, FL. January 23-26, 2000.

[8] Mortari, D., Samaan, M.A., Bruccoleri, C., and Junkins, J.L. "The *Pyramid* Star Pattern Recognition Algorithm," ION *Navigation*, Vol. 51, No. 3, Fall 2004, pp. 171-183.

[9] Bentley, L.J. and Sedgewick, R. "Fast Algorithms for Sorting and Searching Strings," In Proceedings of the 8-th Annual ACM-SIAM Symposium on Discrete Algorithms, pages 360–369, 1997.

[10] Bell, G.I. and Glasstone, S. "Nuclear Reactor Theory." Van Nostrand Reinhold Company, 1970.

[11] Liu, J.S. and Chen, R. "Sequential Monte Carlo Methods for Dynamics Systems," *Journal of the American Statistical Association*, Vol. 93, No. 443, Sep. 1998, pp. 1032-1044.

[12] Carpenter, J., Clifford, P., and Fearnhead, P. "Improved Particle Filter for Nonlinear Problems," IEE *Proc.-Radar, Sonar, Navigation*, Vol. 146, No. 1, Feb. 1999, pp. 2-7.

[13] Kitagawa, G. "Monte Carlo Filter and Smoother for Non-Gaussian Nonlinear State Space Models," *Journal of Computational and Graphical Statistics*, Vol. 5, No. 1, March 1996, pp. 1-25.

[14] Arulampalam, M.S., Maskell, S., Gordon, N., and Clapp, T. "A Tutorial on Particle Filters for Online Nonlinear/Non-Gaussian Bayesian Tracking," IEEE *Transactions on Signal Processing*, Vol. 50, No. 2, February 2002, pp. 174-188.

[15] von Mises, R. *Mathematical Theory of Probability and Statistics*, New York, Academic Press, 1964.

[16] Fisher, R.A, "Dispersion on a Sphere," (1953) Proc. Roy. Soc. London Ser. A., 217: 295-305.

[17] Mardia, K.V. and Jupp, P.E. *Directional Statistics*, John Wiley and Sons Ltd., second edition, 2000.

[18] von Neumann, J. "Various Techniques used in Connection with Random Digits. Monte Carlo Methods," National Bureau Standards, Appl. Math. Ser., Vol. 12, 1951, pp. 36-38.

[19] M. Costello, J. Rogers, "BOOM: A Computer-Aided Engineering Tool for Exterior Ballistics of Smart Projectiles," Army Research Laboratory Contractor Report ARL-CR-670, Aberdeen Proving Ground, MD, June 2011.

[20] R. McCoy "Modern Exterior Ballistics," Schiffer Publishing Ltd., Atglen, PA, 1999, p. 310.